Padrão Strategy: Encapsule responsabilidades criando grupos de classes especializadas

Clube Delphi

Ano 9 :: Edição 126 :: R$ 11,90

DevMedia

ISSN 1517990-7

WordPress
Crie um aplicativo Delphi para monitoramento de feedback de clientes

Expressões Regulares
Conheça a nova biblioteca RegEx nativa do Delphi XE

DataSnap XE, Generics, RTTI e DAO
Transfira e persista objetos em aplicações multicamadas

FINAL BUILDER

Automação de Builds no Delphi XE

Brinde

Comprando esta edição ganhe 2 vídeo-aulas:

- Delphi Prism – Transações
- Delphi Prism – Class Library

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Sumário

Quick Update

06 – Delphi Prism e ADO.NET Entity Framework
[*Guinther Pauli*]

Delphi

Mão na Massa – Boa Ideia

08 – WordPress
Crie um aplicativo Delphi para monitoramento de feedback de clientes
[*Vitor Luiz Rubio*]

Boa Ideia – Novidades

16 – Expressões Regulares
Conheça a nova biblioteca RegEx nativa do Delphi XE
[*Rodrigo Araujo*]

Boas Práticas

Boas Práticas

22 – DataSnap XE, Generics, RTTI e DAO
Transfira e persista objetos em aplicações multicamadas
[*Rodrigo Carreiro Mourão*]

Boas Práticas – Novidades

34 – FinalBuilder
Automação de Build no Delphi XE
[*Daniel Sobrinho Laporte*]

Boas Práticas

42 – Padrão Strategy
Encapsule responsabilidades criando grupos de classes especializadas
[*Rafael Stavarengo*]

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Brinde na web desta edição - 2 vídeos! Confira:

2 Vídeos

• **Curso Delphi Prism: Desenvolvimento Web - Transações**
http://www.devmedia.com.br/post-17825-Transacoes--Banco-de-Dados-Curso-Completo-Delphi-Prism-Desenvolvimento-Web-Parte-66.html

• **Curso Delphi Prism – Dividindo uma aplicação em Class Libraries**
http://www.devmedia.com.br/post-18591-Dividindo-a-aplicacao-em-Class-Libraries-Multicamadas-Curso-Completo-Delphi-Prism-Parte-76.html

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.CD** *Senha:* **GN11FCL**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.CD** Senha: **GN11FCL**

Clube Delphi

Ano 9 - 126ª Edição 2011 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.sobrinho@devmedia.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Jornalista Responsável
Kaline Dolabella - JP24185

Capa e Diagramação
Romulo Araujo - romulo@devmedia.com.br

Revisão
Gregory Monteiro - gregory@clubedelphi.net

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Isabelle Macedo e Uellen Goulart – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Cristiany Queiróz

publicidade@devmedia.com.br

EDITORIAL

Na edição 124 da Revista ClubeDelphi iniciamos nossa cobertura sobre as novidades do Delphi XE, oportunidade em que trouxemos um artigo especial sobre o Subversion, ferramenta agora integrada ao IDE da Embarcadero. Na edição anterior (125) abordamos o CodeSite, a melhor ferramenta de Logging e Debug para Delphi. Esta edição traz a cobertura de mais uma das novidades do novo Delphi XE. O FinalBuilder é a melhor aplicação para automação de Build que existe no mercado, em minha opinião. Em uma situação real, foi o FinalBuilder que reduziu drasticamente o trabalho que tínhamos em um projeto onde existiam mais de 5 mil units e forms. Diariamente o aplicativo compilava centenas de componentes clientes, bibliotecas servidoras DataSnap, formulários, gerava pacotes BPL, os executáveis, enviava tudo para seus devidos lugares (fontes no repositório no StarTeam e aplicações para ambiente de homologação), gerava DOCs atualizados do framework, registrava bibliotecas COM e enviava LOGs de tudo para a equipe. São tarefas rotineiras do dia-a-dia de qualquer desenvolvedor, que podem agora ser totalmente automatizadas. Produtividade na certa para você e sua equipe. Confira um overview da ferramenta na matéria do Daniel Sobrinho.

O Rodrigo Carreiro apresenta um excelente artigo sobre DataSnap, fugindo da tradicional abordagem multicamadas. Aqui são empregadas técnicas avançadas para transferir e persistir objetos em arquiteturas mais modernas, com a utilização de importantes padrões de projeto, como DAO – Data Access Object, além do emprego eficaz de RTTI – Runtime Type Information e Generics. Um excelente guia para criar soluções multicamadas robustas utilizando os mais novos recursos do Delphi XE, da linguagem Delphi e da plataforma DataSnap. Outra novidade do Delphi XE é o suporte nativo a Expressões Regulares, como você vai conferir na matéria do Rodrigo Araujo.

Na sessão Boas Práticas você vai encontrar um artigo do Rafael Stavarengo, abordando o poderoso padrão Strategy, reforçando importantes técnicas da Programação Orientada a Objetos, como Programação para Interfaces, Abstração e Princípio da Responsabilidade Única. O autor utiliza uma das abordagens didáticas mais interessantes já vistas aqui, primeiro apresentará o exemplo com um grande problema de projeto. A seguir, apresenta a solução, afinal, Design Patterns são exatamente isso, soluções prontas para velhos problemas já conhecidos.

Ainda nesta edição, o Vitor Rubio apresenta um artigo Mão na Massa onde cria uma interface Desktop para interagir com o WordPress, um dos principais Content Management System existentes. A solução apresentada é extremamente útil: ajudar a empresa a monitorar feedbacks feitos por clientes da empresa. Boa leitura!

Guinther Pauli
guinther@devmedia.com.br
Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix
Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)
guintherpauli.blogspot.com - twitter.com/guintherpauli

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

DevMedia

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Delphi Prism e ADO.N

O ADO.NET Entity Framework (EF) é a ferramenta de persistência e mapeamento objeto / relacional nativa do .NET Framework, concorrente direto do NHibernate. Ele originalmente apareceu pela primeira vez no Service Pack 1 do Visual Studio 2008 / .NET 3.5. No Visual Studio 2010 / .NET 4.0 ele está na sua versão 2, apesar de normalmente nos referenciarmos a ele por EF4 (versão 4).

O ADO.NET EF permite o desenvolvimento de aplicações para .NET onde o foco principal é a orientação a objetos, sem uma comunicação direta com um servidor de banco de dados relacional, como o SQL Server. É o framework que gera todos os comandos SQL. A última versão permite que o desenvolvedor inicie a criação de um aplicativo pela definição do domínio (entidades), com um diagrama de classes, podendo a seguir gerar todo o esquema para o SGBDR. Essa técnica é conhecida como Model First Development. Se já possui um banco existente, o EF pode fazer a engenharia reversa, gerando entidades no modelo a partir do BD.

Outra característica interessante do EF é a capacidade de suportar tipos complexos, uma novidade da última versão, bem como Lazy Loading, que permite o carregamento de entidades relacionadas conforme utilização. O que considero um dos melhores recursos é o novo suporte a POCO (Plain Old CLR Objects), onde podemos modelar o domínio da aplicação, com as entidades, sem gerar código que esteja atrelado ao framework ou a uma tecnologia em específico. As classes do domínio simplesmente mapeiam a tabela do banco, normalmente sem conter métodos ou derivar de uma classe base para que possam funcionar. Isso dá uma maior flexibilidade e independência ao se trabalhar com o EF. Isso é importante para aplicações em camadas, que preza por baixo acoplamento entre suas partes. O DDD (Domain Driven Design) é a área da Orientação da Objetos que foca na modelagem e arquitetura usando estes princípios.

Com o Delphi Prism XE, Visual Studio 2010 e .NET Framework 4.0, é possível trabalhar com o ADO.NET EF, com algumas limitações. Primeiro, é que não existe, pelo menos até o fechamento desta edição, um template para criar modelos no EF baseados na linguagem Delphi Prism, como ocorre no C#. Porém, devido a integração das linguagens para .NET, é possível usar o template em C# e consumir o modelo a partir do Prism, como veremos aqui.

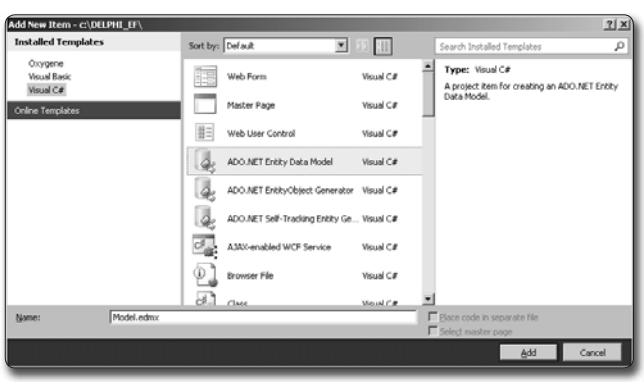


Figura 1. Adicionando um ADO.NET Entity Data Model ao Web Site

Após a criação de um Web Site em Delphi Prism, o primeiro passo é acessar o Solution Explorer e adicionar um item do tipo ADO.NET Entity Data Model, como mostra a **Figura 1**. Um modelo do EF é basicamente um diagrama de classes. Note na esquerda que escolhemos o template C#. Não se preocupe pois não será necessário digitar nenhuma linha de código na linguagem, o Visual Studio irá gerar todo o código necessário para o mapeamento.

A seguir podemos escolher entre iniciar um modelo a partir do zero (Empty model), o que utilizaria a abordagem Model First Development, ou gerá-lo a partir de um banco de dados existente (Generate from database). No exemplo usamos a segunda opção. O próximo passo é escolher o banco de dados a ser utilizado, neste caso consideramos o Northwind do SQL Server.

A **Figura 2** apresenta as tabelas que podem ser mapeadas para entidades. Note que temos uma opção para pluralizar ou singularizar o nome das entidades geradas a partir do banco. Por exemplo, se no banco a tabela se chama Categories, a entidade gerada será Category. Há a opção para incluir no modelo as chaves estrangeiras das tabelas. Apesar de este ser um conceito do modelo relacional, e não da orientação a objetos, pode facilitar a utilização de controles de tela no ASP.NET que relacionam dados.

A **Figura 3** mostra o modelo gerado. Note as duas entidades mapeadas, Category e Product. O relacionamento de Foreign Key no modelo relacional foi mapeado para uma associação. Observe as propriedades de navegação, que permitem por exemplo que você obtenha um objeto do tipo Category e a partir da propriedade Products selecione todos os produtos relacionados com aquela categoria.

Nota

O Delphi já teve frameworks nativos para persistência e mapeamento objeto / relacional. O Bold (Delphi 7) era usado em aplicações Win32. O ECO (Enterprise Core Objects) surgiu no Delphi 8, mas seu uso foi descontinuado na ferramenta.

ET Entity Framework

Guinther Pauli

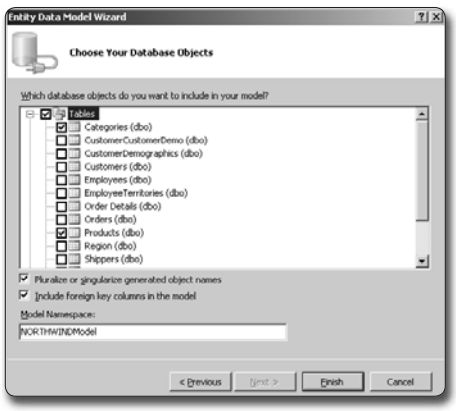


Figura 2. Escolhendo objetos a serem mapeados

Isso é tudo que precisamos para criar o modelo, veja que não foi preciso utilizar C#, apesar do código ter sido gerado nesta linguagem e ela ser tão simples como o Prism. A seguir basta adicionar um Web Form, agora sim em Delphi Prism, colocar um GridView e escolher um Data Source do tipo Entity Data Source. Ele simplesmente aponta para o modelo criado anteriormente. Executando, você terá o resultado da **Figura 4**.

Também é possível executar uma instrução de consulta LINQ – Language Integrated Query – no modelo gerado, derivação que chamamos de LINQ to Entities. Para isso, basta instanciar o modelo e usar uma expressão, como mostrado no código a seguir, que exibe em um GridView todas as categorias:

```
method Default2.Page_Load(sender: Object;
  e: EventArgs);
begin
   var ctx := new NORTHWINDModel.
     NORTHWINDEntities();
   var qry := from C in ctx.Categories
              select C;
   GridView1.DataSource := qry;
   GridView1.DataBind();
end;
```

Agora observe como faríamos para incluir uma nova categoria no banco de

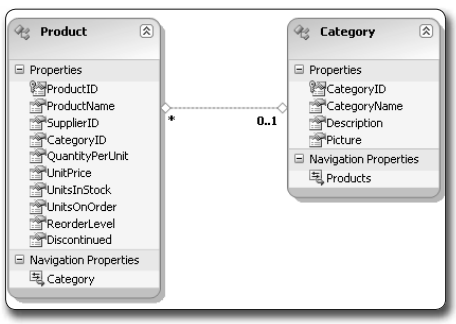


Figura 3. Modelo do Entity Framework

dados. Nada de SQL, nem INSERT, aqui estamos trabalhando com POO. Então criamos um objeto fortemente tipado e pedimos a nossa variável de contexto para adicioná-lo (AddObject), chamando a seguir SaveChanges:

```
method Default2.Button1_Click(sender: System.Object;
e: System.EventArgs);
begin
  var ctx := new NORTHWINDModel.NORTHWINDEntities();
  var Cat := new NORTHWINDModel.Category();
  Cat.CategoryName := "TESTE";
  ctx.AddObject("Categories",Cat);
  ctx.SaveChanges();
end;
```

Nota

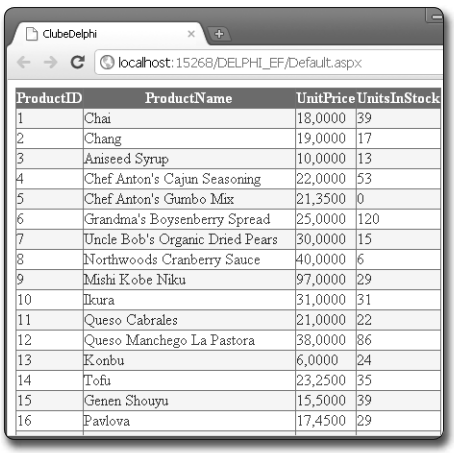
Este exemplo funciona no modelo Web Site, onde podemos ter diferentes partes da aplicação usando diferentes linguagens, em arquivos separados na pasta App_Code. Se fosse usado o modelo Web Application, seria necessário criar um Assembly (Class Library) separado para hospedar o modelo.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



ProductID	ProductName	UnitPrice	UnitsInStock
1	Chai	18,0000	39
2	Chang	19,0000	17
3	Aniseed Syrup	10,0000	13
4	Chef Anton's Cajun Seasoning	22,0000	53
5	Chef Anton's Gumbo Mix	21,3500	0
6	Grandma's Boysenberry Spread	25,0000	120
7	Uncle Bob's Organic Dried Pears	30,0000	15
8	Northwoods Cranberry Sauce	40,0000	6
9	Mishi Kobe Niku	97,0000	29
10	Ikura	31,0000	31
11	Queso Cabrales	21,0000	22
12	Queso Manchego La Pastora	38,0000	86
13	Konbu	6,0000	24
14	Tofu	23,2500	35
15	Genen Shouyu	15,5000	39
16	Pavlova	17,4500	29

Figura 4. Visualizando os dados no Google Chrome

Guinther Pauli
guinther@devmedia.com.br
guinther.pauli@gmail.com
http://twitter.com/guintherpauli
http://gpauli.com

Desenvolve em Delphi há 15 anos, tendo utilizado a ferramenta desde a versão 1, passando pelo Turbo Pascal e Borland Pascal for Windows. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, palestrante em 5 edições da Borland Conference, beta tester de várias versões do Delphi, foi um dos pioneiros na divulgação da ferramenta no país. Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (Visual Studio, C#, ASP.NET), utilizando a plataforma .NET desde a versão beta. Editor das Revistas ClubeDelphi, .NET Magazine e EASY .NET Magazine, professor de Pós-Graduação da UNIVEM-SP, na área .NET. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, MVC, Visual Studio, atuando também em projetos de migração, orientando empresas e desenvolvedores na adoção de melhores práticas, ministra treinamentos em .NET focados em equipes Delphi. Foi arquiteto e desenvolvedor do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

WordPress

Crie um aplicativo Delphi para monit

CMS, ou Content Management System, são sistemas de gerenciamento de conteúdo para um site ou outra mídia qualquer onde um administrador (ou um biblioteconomista) decida tópicos e seções. Dentro desses tópicos e seções podem ser colocados textos de propósito geral: artigos, manuais de produtos, resenhas de produtos, notícias, contos, crônicas e discussões. O texto pode ser enriquecido pela adição de fotos e áudio. Além disso, um artigo pode ou não permitir comentários de visitantes do site. Os comentários podem ou não ser moderados e os usuários podem ou não ser anônimos.

O WordPress é um CMS feito em PHP de código fonte livre, que pode ser estendido por qualquer programador que tenha conhecimentos em PHP / MySql.

Neste artigo será abordado como interagir com um site feito com WordPress da sua empresa, no seu hosting. Não será abordado como interagir com outros sites ou blogs feitos com WordPress. Não será abordada a instalação do WordPress, pois foge do escopo. Como pré-requisito é necessário que haja uma máquina rodando Apache, com PHP e as bibliotecas para acesso ao MySql.

A instalação do WordPress é muito fácil, ao mesmo tempo em que é muito pessoal e varia de acordo com as necessidades da empresa. Cada caso será diferente, mas em todas as instalações, por padrão, será criado um banco de dados MySql (ou usado outro que a empresa já possua) e as tabelas do WordPress serão prefixadas com "wp_" ou outro prefixo que o administrador tenha escolhido para que não se misturem com as outras tabelas que possam existir no banco de dados.

O banco de dados da instalação padrão do WordPress possui onze tabelas, a saber:

- wp_commentmeta
- wp_comments
- wp_links
- wp_options
- wp_postmeta
- wp_posts
- wp_terms
- wp_term_relationships
- wp_term_taxonomy
- wp_usermeta
- wp_users

Porém neste artigo nem todas serão usadas. É possível, e é foco desse artigo, definir outras tabelas neste banco de dados para os sistemas de uma empresa e relacionar essas tabelas com as do próprio CMS. Além dessas tabelas foram incluídas no banco algumas outras para simular o sistema que será integrado com o WordPress. Nada muito complexo. Uma tabela de clientes, onde serão armazenados e-mails/logins dos clientes que pode ser usada em consultas fazendo-se join com a tabela de comentários para verificar se a pessoa que está comentando já é cliente. Uma tabela que será uma lista de palavras características de uma reclamação, a serem monitoradas nos comentários.

Resumo DevMan

De que trata o artigo:

O WordPress é bastante usado para criação de sites de conteúdo dinâmico e propósito geral, mas sua administração tem de ser feita mediante um browser e não pode ser automatizada. Esse artigo mostrará como criar um client em Delphi para o WordPress que pode ser integrado aos seus sistemas.

Para que serve:

Acessando o banco de dados WordPress do site institucional da empresa pode-se rapidamente ver comentários positivos ou negativos sobre o conteúdo e produtos e tirar estatísticas a respeito. Se o WordPress for usado como um blog interno da intranet da sua empresa então pode-se retirar e pesquisar informações de discussões técnicas para a elaboração de uma base de conhecimento.

Em que situação o tema é útil:

Auxiliar o setor estratégico de uma empresa a identificar casos de suporte que mereçam uma atenção diferenciada, detectar reações de um público alvo de um produto, auxiliando também um setor de marketing.

WordPress:

O WordPress é um framework para criação de sites de conteúdo dinâmico e de propósito genérico. Embora a maioria dos sites feitos em WordPress apresentem uma dinâmica de blog, o WordPress pode ser customizado para desenvolvimento de Web Sites diferentes. O sistema de administração do WordPress é, assim como o restante da ferramenta, feito em PHP com MySql e roda a partir de um browser. Ao se desenvolver um cliente Desktop que acessa informações internas do WordPress é possível extrair e cruzar dados de tal forma que possam ser utilizados por um setor de planejamento estratégico.

Nota

Apache – É um servidor HTTP free e open source. Seu principal objetivo é ser seguro, eficiente e extensível, disponibilizando serviços que estejam sintonizados com os mais recentes padrões HTTP

oramento de feedback de clientes

Vitor Luiz Rubio

Por último a tabela que conterá uma relação de comentários que já foram alertados, para que eles não sejam alertados novamente.

Acesso ao MySql

O banco de dados do WordPress apresenta um desafio adicional que talvez nem todos estejam acostumados: a maioria dos campos, senão todos, é not null. Isso significa que não é possível deixar de atribuir algum valor a esses campos. Mesmo assim os campos varchar possuem um valor default caso nada seja atribuído a eles, geralmente uma string vazia de comprimento zero. O grande problema é que no MySql campos memo/text/blob não permitem valor default.

Isso não seria nenhum problema teoricamente, pois bastaria gravar EmptyStr ou '' nesses campos, mas na prática o trio SQLDataset + DatasetProvider + ClientDataset substitui os '' ou até mesmo espaços em branco por null. Em SQL Há várias maneiras de contornar esse problema, uma delas é usar stored procedures, outra é usar comandos de insert montados manualmente.

Neste artigo optou-se por usar o pacote de componentes MySQL DAC, da DEVART, por dois motivos:

1) Não apresenta o problema de substituir '' por null nos campos varchar, embora apresente este mesmo problema nos campos text. Para contornar esse problema pode ser gravado espaço em branco no lugar de string vazia. No MySql DAC o espaço em branco é aceito normalmente. Outra maneira mais elegante de resolver esse problema é usar a função coalesce nos parâmetros do tipo text das instruções SQL tanto de update como de insert;

2) É um dos mais rápidos para se trabalhar com MYSQL, mais rápido até do que o DBExpress, inclusive é possível trabalhar com ele pela web sem dificuldade alguma.

Nota do DevMan

Recentemente a Microsoft adotou o WordPress como mecanismo de blog padrão das redes Windows Live e colocou no site uma ferramenta para que todo conteúdo possa ser migrado para o WordPress.

Mesmo usando MySql DAC seria muito mais interessante e elegante utilizar stored procedures. Com elas é possível inicializar um parâmetro com '' ou atribuir-lhe '' se ele for passado como null, e assim não correr o risco de tentar gravar null em um campo not null.

O pacote MySql DAC é pago, mas um Trial de 60 dias pode ser baixado do site da Devart, o link pode ser conferido na seção de links desse artigo. Além dos componentes para acesso ao MySql há também outros para Oracle, SQL Server e até um pacote completo com os principais bancos de dados do mercado. Seu funcionamento é de uma maneira parecida com os componentes TQuery ou TIBQuery antigos, sem a necessidade de um DatasetProvider ou um ClientDataset, e é possível montar as instruções SQL de insert, update, delete etc. de maneira flexível e personalizada.

Estrutura do banco de dados

Para explorar a estrutura do banco de dados foi usado o ER Studio Data Architect XE, com a opção de engenharia reversa, que cria o diagrama de tabelas do WordPress através do banco de dados. Mesmo não possuindo acesso nativo ao MySQL, foi possível realizar a conexão utilizando um driver ODBC. A **Figura 1** exibe o diagrama gerado.

Nota

A função coalesce aceita uma lista de parâmetros de tamanho variável e retornará o primeiro parâmetro não nulo. Usar coalesce em um parâmetro de entrada de uma query de insert ou update é uma saída para o problema de string vazia/null, porque usando a função coalesce(:parâmetro,''), caso o Delphi trocasse a string vazia '' entrada pelo usuário por null, a função coalesce ficaria coalesce(null, '') e retornaria '', o primeiro parâmetro não null.

Nota

É uma boa alternativa criar uma constante chamada 'ESPACO' para colocar nela o caractere de espaço ou até mesmo o caractere "nonblank space" (u0160 na tabela unicode). Com isso o código fica mais legível, pois evita confundir-se espaço ' ' com string vazia ''. Além disso se numa versão posterior esse comportamento que transforma espaços em null for alterado basta apenas alterar a constante ESPACO para string vazia.

Nota

O TMyQuery, assim como outros componentes de query tradicionais como TQuery e TIbQuery permite a edição de scripts personalizados para as operações de insert, delete, update e select. Neste exemplo o select foi feito com select * from wp_posts para efeito de simplicidade, mas não é uma boa prática, o correto seria relacionar os campos a serem obtidos. O diferencial do TMyQuery é que, além de ter um Component Editor para gerar automaticamente as instruções SQL, ele também pode gerar instruções para execução de stored procedures.

Nota

O banco de dados usado nesse artigo possui login root e senha "123456" e o nome do banco é WordPress. Para realizar a conexão foi usado o nome de servidor "localhost" porque o banco de dados está no próprio computador de desenvolvimento.

Nota

Baseando-se nos nomes das entidades e chaves, o ER Studio cria um relacionamento que muitas vezes não existe no banco físico. É preciso então observar a estrutura real do banco de dados e corrigir o diagrama gerado.

Nota do DevMan

Open Database Connectivity (ODBC) foi desenvolvido pela Microsoft como uma interface que garante o acesso a dados aos mais diversos bancos relacionais e não relacionais. Ele disponibiliza uma API neutra, que não está relacionada com nenhum fornecedor de banco de dados. Um ponto forte do ODBC é que ele diminui a necessidade de se aprender vários frameworks de acesso a dados, já que ele fornece uma interface universal. Isso permite que aplicações acessem, visualizem e modifiquem informações de diversos bancos de dados.

Alterando a estrutura do banco de dados

Para simular a existência de um sistema que se integrará ao WordPress algumas tabelas são adicionadas à estrutura conhecida. A tabela monitor_reclamacao para salvar uma lista de palavras a serem monitoradas, que causará o envio de alertas ao atendimento. A tabela comments_alertados salva uma lista de comentários que já foram alertados ou classificados para alerta, assim um comentário não será enviado mais de uma vez para o atendimento. A tabela clientes, com campos como CPF e e-mail, representa o cadastro de clientes que a empresa já possui.

As **Tabelas 1, 2** e **3** mostram a estrutura (campos e tipos) das tabelas adicionadas ao banco do WordPress.

Não há sistema ou interface para inserção de dados na tabela de clientes, pois não é o foco desse artigo, por isso uma pequena massa de dados de teste é adicionada na tabela clientes (**Tabela 4**) e em Monitor_Reclamacao (**Tabela 5**).

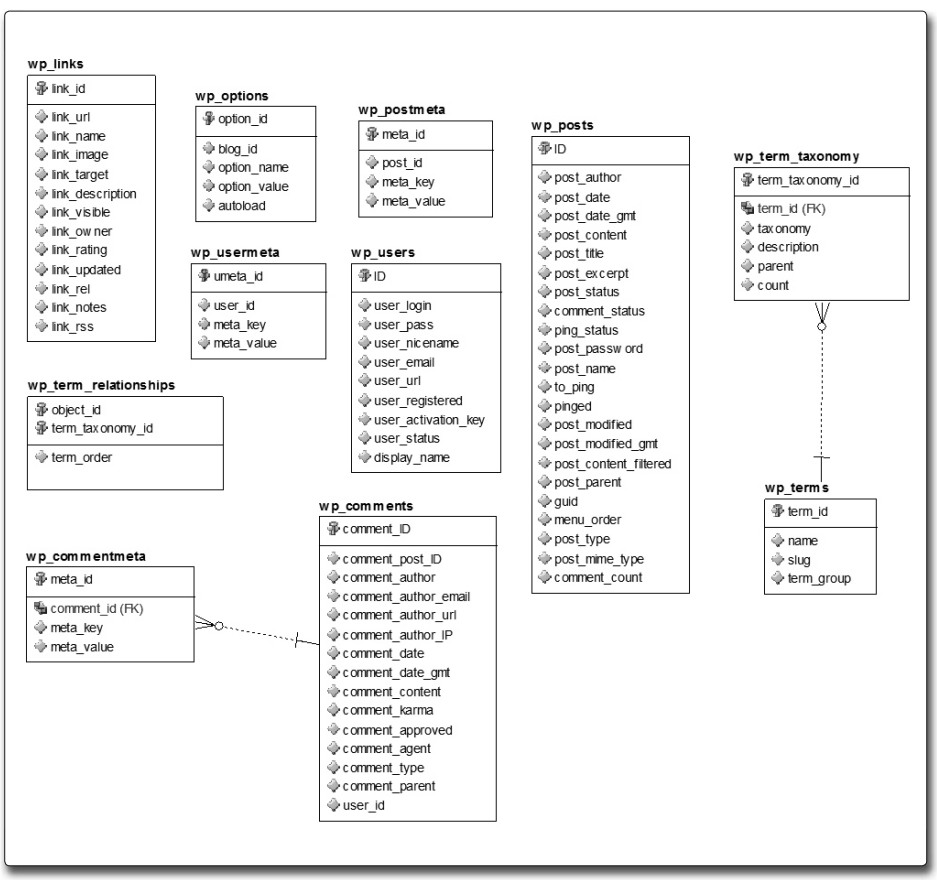


Figura 1. Diagrama do banco do WordPress

Manter uma lista de palavras e/ou expressões em uma tabela para monitorar todos os comentários buscando pelas palavras listadas não é a maneira mais elegante. O ideal seria usar full-text search, ou separar as palavras em tipos, classes, quanto à sua semântica e sua composição (radicais, sufixos, prefixos). Há algoritmos complexos de inteligência artificial para se tratar palavras/língua, mas o objetivo deste artigo é apenas didático, por isso as palavras monitoradas serão armazenadas em uma única tabela.

A tabela Comments_Alertados é preenchida com o uso do sistema e serve para que um comentário que já foi alertado não o seja novamente. Entenda-se por alertado a execução de um evento qualquer, seja um aviso em tela, envio de um e-mail para SAC, marketing ou moderação ou mesmo uma combinação entre esses. Para combinação a tabela poderia ter, além do ID do comentário, flags booleanas para marcar se já enviou e-mail, sms e assim por diante.

Adicionando um post

Por motivos de simplicidade este artigo segue a metodologia RAD (Rapid Application Development) que pode ou não incorporar elementos de POO, mas não os têm como foco. Aplicações RAD são mais orientadas a eventos e têm a característica de se colocarem componentes de acesso a dados e de interface com o usuário em um mesmo formulário, conectá-los e programar as regras de negócio e validações nos eventos dos componentes.

Para adicionar um post é necessário criar um registro na tabela wp_posts. O conteúdo é o campo post_content. Os demais campos têm nomes bem sugestivos. Os valores possíveis para post_type são “post”, “page” e “revision”. Pages são páginas de conteúdo fixo e podem ser ligadas com menus do site, enquanto “posts” são como entradas de um blog, geralmente mostradas na home.

Revisions são históricos de alterações. Cada edição de um post gera automaticamente na tabela wp_posts um post novo com o conteúdo atual. Assim é possível ter um histórico de tudo o que foi alterado,

Nota do DevMan

Full-Text Search
É uma forma especial de se buscar por um texto no meio de um banco de dados. Consultas por texto não são eficientes no meio relacional porque geralmente causam uma varredura nas tabelas envolvidas na busca, já uma consulta full-text search é otimizada para isso, ela se prepara para as buscas armazenando as palavras em um formato eficiente. Esse é o chamado indexing ou, indexação.

Indexing
É o trabalho de construir uma estrutura que torne a pesquisa rápida e fácil. Cada implementador segue um padrão de formato que difere do padrão relacional.

Searching
É processo de obter o texto a ser localizado e transformar em parâmetros para busca. A primeira coisa que um gerenciador full-text faz é analisar o texto para gerar o índice de busca. O passo seguinte é recuperar do index todos os registros que estão relacionados com as palavras digitadas na busca. Nesse momento o registro em si não é carregado, apenas é recuperado o id de cada registro. De posse dos ids dos registros, é possível aí sim realizar uma consulta relacional utilizando como busca esses ids encontrados.

e até um sistema de versionamento do conteúdo.

Quando uma revisão é criada o campo post_parent é preenchido com o número do post que foi revisado. Trata-se de uma tabela que possui um campo que é chave estrangeira da própria tabela. Uma tabela auto-referenciada, num esquema de árvore, de um único nível nesse caso. Sendo assim, ao editar um post pode-se procurar por todos os posts que tenham como post_parent o código do post sendo editado para se ter uma lista de revisões. Uma query pode ser feita usando-se post_parent como filtro e os resultados obtidos podem ser usados numa espécie de "botão desfazer" para se voltar a uma revisão anterior.

Para criar um post é necessário escolher um autor e preencher o campo post_author com o código deste. Ele deve estar previamente cadastrado na tabela wp_users. Há duas maneiras de se fazer isso: ou o usuário do client entra com login e senha ao inicializar o sistema e seu login do sistema é associado a um login de autor pelo e-mail ou o usuário simplesmente visualiza uma janela ou lookup e seleciona um de vários autores ou pseudônimos existentes. Tudo dependerá da regra de negócio do sistema.

A **Figura 2** mostra como foi montada a tela para adição de posts.

Campo	Tipo
CodCli(pk)	int
NomeCli	Varchar(80)
CpfCli	Varchar(11)
EmailCli	Varchar(80)

Tabela 1. Clientes

Campo	Tipo
Palavra (pk)	Varchar(100)

Tabela 2. Monitor_Reclamacao

Campo	Tipo
CommentId (fk para a pk de wp_comments)	int

Tabela 3. Comments_Alertados

CodCli	NomeCli	CpfCli	EmailCli
1	João	56371066730	joao@provedor.com.br
2	Maria	91401773516	maria@provedor.com.br
3	Pedro	80328717215	pedro@provedor.com.br

Tabela 4. Clientes

Como pode ser visto na **Figura 2** o formulário foi montado de forma que o campo Autor fosse um TDBLookupComboBox. Poderia ser também um TDBLookupListBox ou até um grid com DataSource e Dataset em um formulário separado a ser exibido com showmodal, apenas para se obter o código do autor. Para usar o TDBLookupComboBox colocou-se no formulário um outro TMyQuery selecionando os dados da tabela wp_posts e ele foi setado como lookup source do TDBLookupComboBox. Ao se escolher um valor no TDBLookupComboBox automaticamente o código do autor escolhido vai para o campo correspondente do qryPosts. Um ponto interessante é que ao clicar no botão Novo é preciso setar os campos com uma string vazia, justamente por conta de serem not-null no banco de dados e outros valores padrão também são definidos (**Listagem 1**).

A edição de um post pode ser feita clicando-se no registro desejado no DBGrid e alterando as informações necessárias. Ao salvar, seja um post novo ou um alterado, ele poderá ser visto na página online.

Adicionando um comentário

Adicionar um comentário é muito parecido com adicionar um post, mas a tabela de posts é a tabela mestre e comentários é a detalhe, por isso o menu ou botão para

Palavra
Reclamar
Não gostei
Má qualidade
Palavrão

Tabela 5. Monitor_Reclamacao

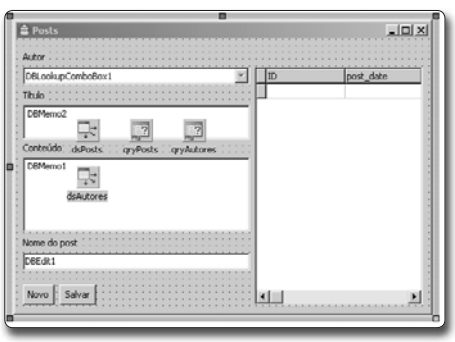


Figura 2. Montagem do formulário

se adicionar um comentário a um post deve ficar na janela onde se visualiza/edita o post.

A maior vantagem de se acessar a tabela de comentários é procurar por e-mails dos usuários que comentam e relacioná-los com os e-mails dos usuários do seu sistema. Se o WordPress for instalado no mesmo banco de dados que o banco do sistema da empresa é possível até criar querys com joins entre a tabela de clientes e a tabela de comentários, ligando-as pelo e-mail. Isso

permite a empresa ter um feedback muito rápido das reações dos usuários a um novo post, por exemplo sobre um novo produto. Isso pode auxiliar a criação de um atendimento diferenciado ao cliente.

Outro motivo de se acessar essa tabela é o mail marketing. Cada e-mail pode ser um prospectivo cliente, principalmente em se tratando de e-commerces.

É possível armazenar em uma tabela do sistema pré-cadastros que podem ser complementados quando um cliente resolver se cadastrar e se tornar ativo. O nome e e-mail dele já existirão, faltando apenas CPF e endereço ou outras informações de contato pertinentes ao sistema da empresa.

Ainda outro motivo é o próprio gerenciamento de comentários. Caso precisem passar por moderação, é necessário de tempos em tempos acessar o admin do site, entrar em um determinado post e verificar se tem novos comentários a moderar. De tempos em tempos é necessário atualizar o browser para ver a entrada de novos comentários.

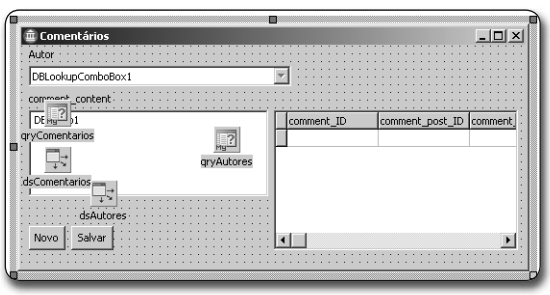


Figura 3. Formulário de comentários

Listagem 1. Valores padrão

```
qryPostsID.AsInteger := 0;
qryPostspost_date.AsDateTime := now;
qryPostspost_date_gmt.AsDateTime := now;
qryPostspost_status.AsString := 'publish';
qryPostscomment_status.AsString := 'open';
qryPostsping_status.AsString := 'open';
qryPostspost_modified.AsDateTime := now;
qryPostspost_modified_gmt.AsDateTime := now;
qryPostspost_parent.AsInteger := 0;
qryPostsmenu_order.AsInteger := 0;
qryPostspost_type.AsString := 'post';
qryPostscomment_count.AsInteger := 0;
qryPostspost_password.AsString := EmptyStr;
qryPostsguid.AsString := EmptyStr;
qryPostspost_mime_type.AsString := EmptyStr;
qryPostspost_excerpt.AsString := EmptyStr;
qryPoststo_ping.AsString := EmptyStr;
qryPostspost_content_filtered.AsString := EmptyStr;
qryPostspinged.AsString := EmptyStr;
```

Usando um cliente desktop e um timer é possível encontrar, moderar e até responder um comentário quase que reativamente à sua inserção no post. Isto significa que imediatamente é possível dar uma resposta pública ao cliente, particular via e-mail ou até mesmo efetuar-se uma ligação para o cliente.

Além de monitorar dúvidas pode-se monitorar reclamações. Embora o WordPress tenha mecanismos muito bons de detecção de spam, lista negra de palavras (para bloqueio de comentários com palavras de baixo calão) e moderação, não é interessante bloquear certos comentários sem se ter noção do porquê eles foram bloqueados. Novamente, comentários de clientes insatisfeitos e reclamações podem disparar um alarme (até mesmo sonoro) no sistema para que a equipe do SAC possa prontamente entrar em contato com o cliente.

A ideia básica é abrir o formulário de comentários de maneira modal já informando qual o código do post escolhido. Assim será permitido adicionar ou moderar apenas comentários relativos a este post.

Para que isso seja possível foi usado um parâmetro na query que traz os comentários.

Isso também poderia ser feito com dois TMyQuerys aninhados de maneira mestre / detalhe, ou colocando-se no formulário de comentários um lookup para posts ou ainda fazendo o contrário: a tela de comentários abrindo a tela de posts para escolher-se de qual post se fará o comentário.

A maneira mais simples encontrada na elaboração deste artigo foi colocar um menu popup no DBGrid do frmPosts com o item "Comentarios". Clicando-se neste item é possível chamar a janela de comentários modal e adicionar um comentário (**Figura 3**).

O formulário de comentários possui o mesmo problema que o de posts: precisa de um autor e tem vários campos text que não podem ser nulos (uso de coalesce). Aqui já pode ser percebido um ponto de melhoria futura: isolar qryAutores em um formulário, DataModule ou classe separada a ser instanciado sob demanda.

Não confundir o campo comment_author, pois este é do tipo tinytext e deve aceitar strings, para o usuário por seu nome ou apelido. O campo user_id não pode ser nulo, mas pode ser zero caso o usuário seja anônimo, ou pode ser um código de usuário válido, podendo ser o código do autor do post ou de um dos administradores. Por isso o DataField do lookup neste formulário é o user_id, por padrão "1", que é o ID do admin, é melhor do que zero quando for um comentário feito através do Delphi e não através do site.

Outro ponto de melhoria é validar o que está sendo preenchido antes de chamar o post, assim não há motivos para disparar exceptions por falta de dados preenchidos. Isto pode exibir, desnecessariamente, detalhes da arquitetura do sistema.

Na **Listagem 2** os valores padrões dos campos são definidos logo antes do post em vez de depois do insert, pois antes de clicar no Salvar o usuário pode mudar várias vezes o autor selecionado. Além disso, o usuário pode sair digitando um comentário para um post que ainda não possui nenhum sem clicar no Novo, e isso geraria um erro, por isso, antes de post, nenhum campo ficará null, mesmo que o insert tenha sido automático.

É mais elegante colocar esse tipo de código nos eventos AfterInsert e BeforePost, mas como este projeto é muito simples o código fica mais claro de se entender nos próprios botões que disparam as ações.

Outro ponto digno de nota é a propriedade comment_post_ID. Esta propriedade não lê e grava os valores diretamente no campo privado Fcomment_post_ID, mas usa acessores (Getters e Setters) para isso. O Getcomment_post_ID apenas retorna o Fcomment_post_ID, mas o Setcomment_post_ID faz algo mais: preenche o parâmetro da consulta, abre a consulta e exibe no caption o código do post.

O comentário tem uma propriedade parent. Isso porque pode existir um comentário em resposta a outro comentário, aninhado. Para criar uma resposta bastaria fazer na janela de comentário a mesma

coisa que foi feita na janela de posts, mas desta vez abrindo outra instância da janela de comentário e passando não só o ID do post, mas o ID do comentário "pai" também.

Por padrão os comentários são moderados. Isso pode ser mudado nas configurações do admin da ferramenta. Para criar comentários pré-aprovados ou aprová-los pelo Delphi o campo comment_approved deve conter o valor "1". Ao salvar comentários ele estarão disponíveis nos posts online. Contudo um erro poder ser notado. A contagem de comentários está incorreta.

Para sanar esse problema o post deve ser editado ao ser fechada a janela de comentários e o total de comentários deve ser gravado no campo comment_count. A **Listagem 3** apresenta essa alteração.

Listagem 2. Formulário de comentários

```
qryComentarioscomment_post_ID.AsInteger := Fcomment_post_ID;
qryComentarioscomment_date.AsDateTime := now;
qryComentarioscomment_date_gmt.AsDateTime := now;
qryComentarioscomment_karma.AsInteger := 0;
qryComentarioscomment_approved.AsInteger := 1;
qryComentarioscomment_agent.AsString := '';
qryComentarioscomment_type.AsString := '';
qryComentarioscomment_parent.AsInteger := 0;
//nome do usuário da tabela de autores escolhido no lookup
qryComentarioscomment_author.AsString := qryAutoresdisplay_name.AsString;
//e-mail do usuário
qryComentarioscomment_author_email.AsString := qryAutoresuser_email.AsString;
//url do site/blog da empresa/admin
qryComentarioscomment_author_url.AsString := qryAutoresuser_url.AsString;
//colocando o ip de localhost/loopback para fins de teste
qryComentarioscomment_author_IP.AsString := '127.0.0.1';
qryComentarios.Post;
```

Listagem 3. Salvando a contagem de comentários

```
with TfrmComentarios.Create(nil) do
try
 comment_post_ID := qryPostsID.AsInteger;
 ShowModal;
 qryPosts.Edit;
 qryPostscomment_count.AsInteger := qryComentarios.RecordCount;
 qryPosts.Post;
finally
 Release;
end;
```

Monitorando comentários

É perfeitamente viável criar tabelas de listas de palavras a serem monitoradas e criar um programa que de tempos em tempos monitore os comentários dos posts. O WordPress possui a funcionalidade de bloquear comentários em posts antigos, o padrão é 14 dias mas isso pode ser alterado. Sendo assim o monitor de comentários não tem uma quantidade muito grande de posts a monitorar, apenas os liberados para comentários.

Para construir esse monitor é necessário uma janela bem parecida com o formulário de comentários, mas a consulta trará todos de uma determinada faixa de datas, que não estejam em comments_alertados e que estejam aguardando moderação (esta parte é opcional). Sendo assim poucos comentários serão trazidos. O formulário conterá um memo para registrar as ocorrências e um timer que a cada 30 segundos buscará novos comentários e pesquisará neles pelas palavras de monitor_reclamacao (**Figura 4**).

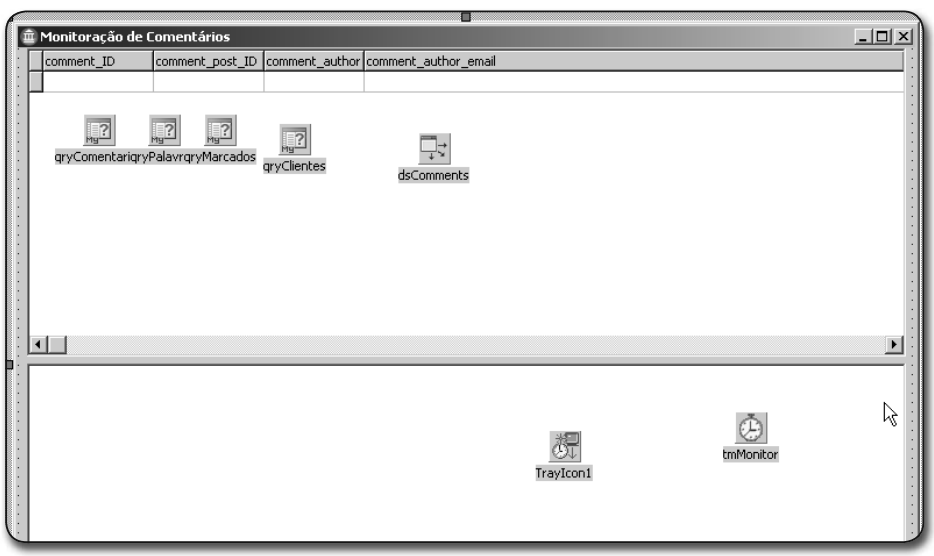


Figura 4. Monitor de comentários

A tabela monitor_reclamacao possui vários registros. Para que não se faça várias consultas por minuto, primeiro todos os comentários serão trazidos e depois, em loop se varre a lista de palavras. É verificado se os comments possuem essas palavras. Caso possuam é feito um registro no log, simulando o e-mail/alerta e é gravado o ID do comentário na tabela comments_alertados, para que não se dispare alerta para o mesmo comment.

Esta janela não permite a edição dos registros (mas pode permitir a aprovação) e deverá iniciar junto com o projeto e ter um ícone na System Tray alertando sua atividade. A lógica do monitor de comentários é simples: a cada intervalo do timer são abertos no qryComentarios todos os comentários não moderados num intervalo de 31 dias e que não foram monitorados anteriormente. Depois de recuperados esses comentários, é varrida a lista de palavras (qryPalavras) e usado um filter com like para procurar um comentário com as palavras varridas.

Se uma palavra for encontrada o comentário será marcado e um registro no log será exibido. O log feito aqui não é persistente, mas é bom que seja. Além do log é interessante implementar envios de e-mail e alertas sonoros conforme o caso.

Depois de verificada a existência da palavra, é verificado se o usuário que comentou em questão é ou não cliente. Se for cliente será concatenada a mensagem "e ainda por cima é cliente, muito cuidado", caso contrário será concatenada a mensagem "o usuário está chateado, mas não foi identificado como cliente" mostrando que ele pode ser pelo menos um prospectivo cliente, mas está insatisfeito. Pode também ser um cliente indireto, que comprou seu produto em uma loja e não diretamente da sua empresa.

Um filter é usado no lugar de locate porque locate busca apenas o início das strings. Além disso o filter aplica o filtro nas informações que já estão no cache do Dataset e não faz uma consulta SQL novamente. Sendo assim, a consulta para trazer a lista de palavras ocorre apenas quando o formulário é criado, a consulta de comentários é feita a cada ciclo do timer, mas não é repetida ao longo do loop de varredura das palavras.

Além disso há ainda a qryClientes, para verificar se o e-mail do "comentante" é de um cliente. Esta query aceita um parâmetro do tipo string que é o e-mail. A outra query, qryMarcados, serve para marcar comentários já monitorados. A **Figura 5** mostra o monitor de comentários em ação.

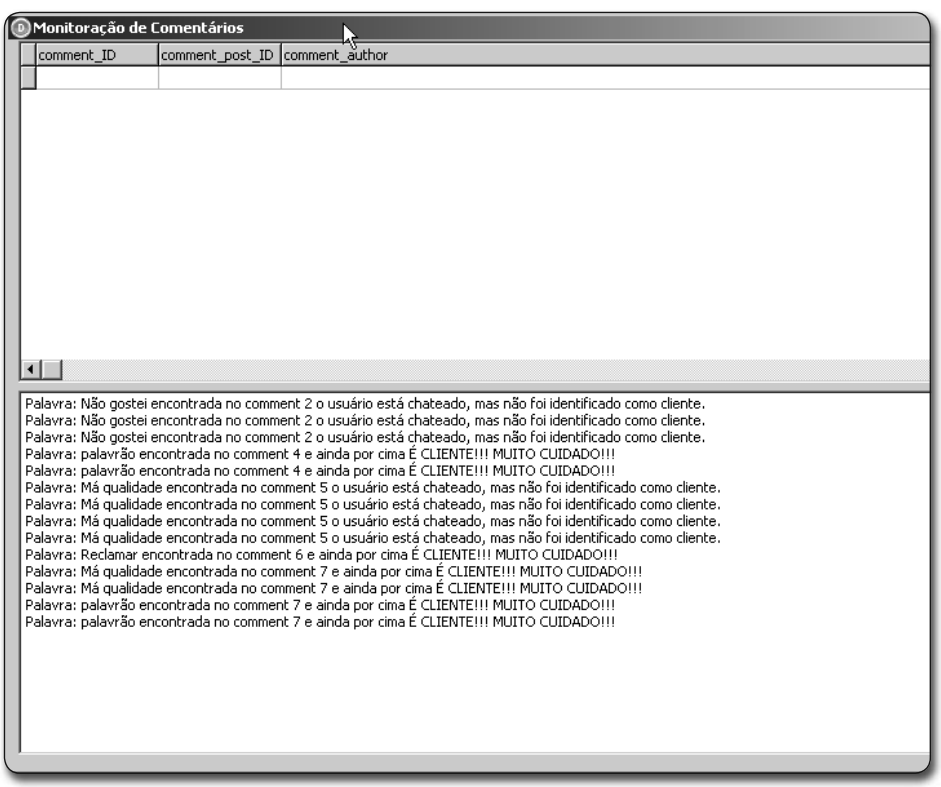


Figura 5. Monitor de comentários em ação

Nota do DevMan

DAO ou Data Access Objects são classes que centralizam em si as operações básicas de inclusão, atualização, exclusão e localização de dados em um banco de dados. Compõem a chamada camada de acesso a dados em um sistema.

Nota

É dito que um código é thread safe quando se prova que ele funciona corretamente quando executado simultaneamente por várias threads.

Nota

Para evitar que os comments dos próprios administradores sejam monitorados eles podem ter seus IDs previamente registrados na tabela comments_alertados.

O sistema diferenciará quando os usuários são clientes ou não para prioridade no atendimento. Por padrão do WordPress os comentários aguardam moderação, então não serão publicados os comentários negativos, apenas monitorados. Eles podem ser publicados depois de editados e respondidos ao cliente.

Devido ao intervalo de tempo muito curto algumas mensagens do log se repetiram de maneira idêntica com exceção do comentário 7 que tinha realmente duas palavras a serem filtradas, palavrão no nome e má qualidade no content.

Isso pode ser minimizado com o aumento do intervalo do timer e totalmente resolvido caso o campo commentId da tabela comments_alertados seja único e não se tente inserir nele um comment que já foi inserido antes (para não disparar exception de key violation).

Outra medida de segurança thread safe seria criar e destruir uma instância de TMyQuery diferente para trazer os resultados dos comentários a cada ciclo do timer em vez de se usar um componente público na form. Sempre surgirão oportunidades de se melhorar códigos ou regras de negócio e boas práticas nunca serão demais.

Expandindo horizontes

Esse artigo segue a metodologia RAD por questões de didática e simplicidade. É sensato considerar a criação de classes de negócio para cada entidade do WordPress (ou qualquer outro produto que venha a se integrar com alguma aplicação) e outra camada de classes para as entidades DAO ou de persistência.

Outra arquitetura que pode ser considerada é a instalação de um servidor DataSnap no mesmo servidor que o Apache/MySql/WordPress ou em outro na mesma rede e a implantação de um modelo em múltiplas camadas. Com a difusão e uso da Web hoje, as oportunidades de negócio estão na Web, seja nos blogs, fóruns ou redes sociais. Dessa forma todo tipo de integração com ambientes ou aplicações na plataforma Web é válido.

Caso a empresa em questão tenha um fórum para discussão sobre uso dos seus produtos vale a mesma técnica mostrada

neste artigo para blogs. Se for possível obter o e-mail da pessoa que pergunta, responde, reclama ou de qualquer modo se relaciona com o site, cruzando este e-mail com a base de clientes é possível criar um atendimento personalizado e imediato ao cliente. Isso diminui o impacto negativo que um cliente insatisfeito pode criar na Web e gera uma boa impressão, de bom atendimento. Em longo prazo o impacto positivo será bem maior.

Se os e-mails coletados não forem clientes eles podem ser prospectivos clientes. Pode-se mandar a eles mala direta, contanto que haja mecanismos para eles cancelarem o envio, caso contrário torna-se spam. Mesmo o cliente não querendo mais receber mala direta, um simples clique no link para remover os e-mails da campanha pode mostrar que o cliente de fato existe, é uma pessoa real, e que ele abre os seus e-mails, mesmo que seja para clicar no cancelamento.

Há vários produtos open-source e livres na internet para rápida criação de sites que podem ser integrados com um ERP, AMS (Account Management System) ou um CRM, desde lojas virtuais a mecanismos de fórum e blogs.

Conclusão

O WordPress é apenas um dos vários frameworks que podem ser usados para construção de sites. Foi escolhido para a elaboração deste artigo porque além de vários usuários já terem familiaridade com o WordPress, sendo ele o mecanismo oficial de blogs da Microsoft hoje, ele é mais flexível do que algumas ferramentas proprietárias, é open-source e possui livros em português.

Outro motivo que levou à escolha do WordPress foi o fato de ele ter sido feito em PHP, linguagem também livre e amplamente conhecida. Pode-se inclusive usar o RadPHP da Embarcadero para se criar novas funcionalidades para o WordPress.

Espera-se com esse artigo que o leitor esteja apto a inserir informações em sites WordPress diretamente pelo banco de dados ou extrair informações dos sites para campanhas de marketing, levantamentos estatísticos e monitoramento de conteúdos e menções.

Vitor Luiz Rubio
vitorrubio@gmail.com
http://www.vitorrubio.blogspot.com
http://twitter.com/vitorrubio
http://vitorrubio.WordPress .com/
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP. Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais. Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

MySql Connector
http://dev.mysql.com/downloads/mirror.php?id=396600#mirrors

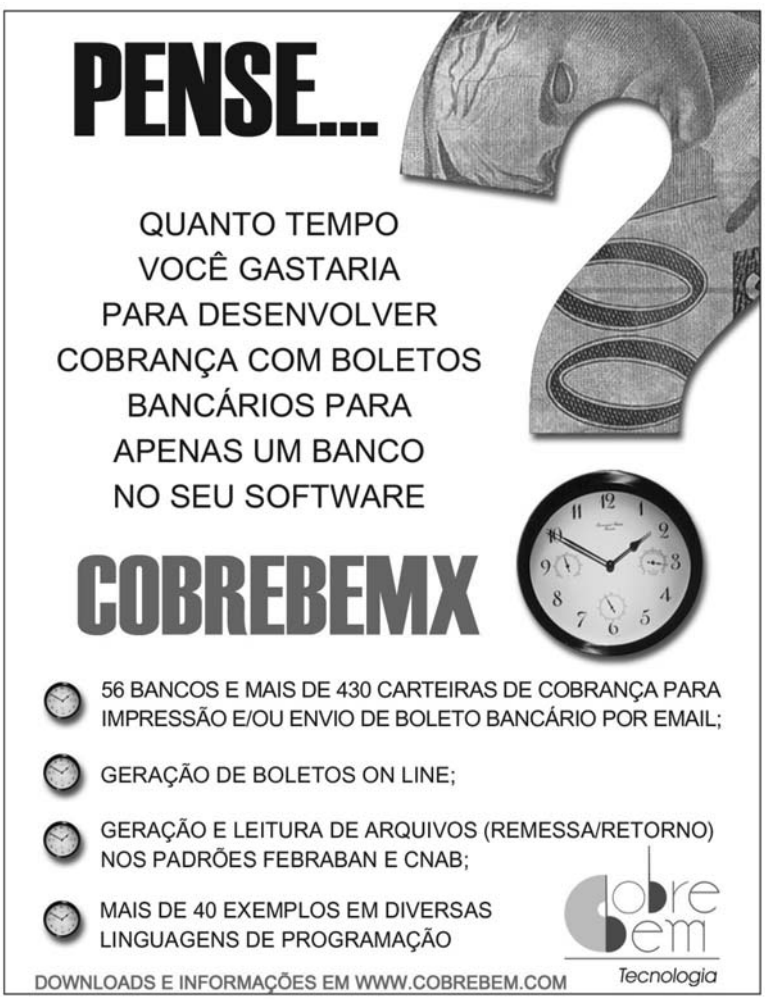
My Sql DAC
http://www.devart.com/mydac/

WordPress
http://wordpress.org/

Apache
http://httpd.apache.org/

PHP
http://www.php.net/

MySql
http://www.mysql.com/



Expressões Regulares

Conheça a nova biblioteca RegEx

As expressões regulares surgiram em meados de 1943, quando dois neurologistas, MacCulloc e Pitts, publicaram um estudo teórico indicando o funcionamento dos neurônios, onde sua teoria estava centrada em autômatos finitos e em teorias de linguagens formais. Já na década de 1950, um matemático chamado Stephen Kleene realizou um estudo utilizando tais modelos usando uma notação matemática que ele chamou de grupos regulares, posteriormente formando a álgebra de Kleene. Mas somente em meados de 1968 ocorreu o encontro com a computação, quando Ken Thompson utilizou a notação da álgebra em um algoritmo de busca utilizando um editor de textos chamado "qed", que posteriormente se transformou em "Ed", editor de textos padrão de sistemas Unix. Neste editor de textos era possível realizar buscas utilizando expressões regulares através de um comando "g" e um comando de impressão chamado de "p", representando algo semelhante a Global Regular Expression Print, como evolução, dando origem ao aplicativo "grep". Em 1986 surgiu um conjunto de bibliotecas pioneiras escritas na linguagem de programação C chamadas de regex que descrevia o tratamento de expressões regulares para aplicativos.

Neste aspecto, realizando um comparativo a edição de um texto, para expressar as ideias de um determinado assunto, ele pode ser escrito de forma diferente por autores distintos, ou seja, cada autor terá sua maneira de expor seus argumentos relacionados ao tema indicado. Apesar disso, o texto possuirá regras básicas relacionadas à forma como ele será redigido. Seguirá uma sequência lógica de raciocínio, onde apresentará uma introdução, também chamada de início, um desenvolvimento, apresentando as ideias básicas do assunto e suas argumentações teóricas e o final representado pelas conclusões.

De maneira semelhante, uma expressão regular pode ser analisada, na qual ela poderá conter um início, meio e fim, respeitando determinados critérios de criação que serão observados na avaliação da expressão que será informada. Uma expressão regular é representada por um conjunto de caracteres, podendo conter números e símbolos agrupados em determinada ordem pré-definida representando uma expressão de validação de uma regra, onde uma entrada de dados será analisada nessa expressão, indicando se é um valor que satisfaz suas condições, ou seja, um valor válido para a expressão.

As expressões regulares são planejadas e escritas a partir de pequenos blocos de código. Cada bloco individual é bastante simples, possuindo uma sequência lógica distinta. Os blocos podem se agrupar de diversas maneiras para atingir um objetivo específico.

Elas podem ser utilizadas com diversos propósitos, para manipular um conjunto de fluxo de dados de entrada, realizando testes condicionais, observando as regras definidas, para produzir o resultado final desejado ou, então, informarão que a análise não foi válida, pois o fluxo de dados estava em um formato incorreto.

É um método poderoso e flexível para utilizar um determinado padrão de texto, ou seja, a partir da composição de símbolos, da utilização de caracteres com funções especiais (meta-caracteres) e caracteres literais, compondo uma expressão, que

Resumo DevMan

De que trata o artigo:

O artigo trata de expressões regulares, demonstrando o seu surgimento, seus conceitos básicos, principais meta-caracteres e sua utilização na prática, onde é utilizado o Delphi XE para a reprodução de exemplos simples de expressões.

Para que serve:

As expressões regulares servem para criar padrões de validações de dados específicos unindo a flexibilidade com elegância. Utilizando-as é possível criar cadeias ou padrões de caracteres e/ou palavras possibilitando uma validação de informação mais "inteligente".

Em que situação o tema é útil:

Observando os princípios básicos de expressões regulares pode-se analisar e validar sequências. Além disto, permite um ganho de produtividade e uma melhor qualidade de informações analisadas, poupando bastante trabalho principalmente na avaliação, pesquisa e substituição de textos. Um exemplo muito comum é a validação de e-mails.

Expressões Regulares:

Neste artigo será abordado um pouco da história das expressões regulares desde o seu surgimento à sua aplicação cotidiana. Conheceremos os principais conceitos e operadores utilizados para criar e entender expressões regulares, e veremos a aplicabilidade do tema na nova versão do Delphi XE, recentemente lançando pela Embarcadero e que possui um suporte nativo para a utilização das expressões regulares.

nativa do Delphi XE

Rodrigo Araujo

será interpretada como regra para validação dos dados informados. As expressões regulares são projetadas e otimizadas para a manipulação de texto, são compostas por dois tipos básicos: caracteres de texto literal e caracteres especiais, onde a adição dos dois tipos permite às expressões regulares grande capacidade de processamento e análise de textos.

Uma expressão regular, também chamada de RE ou REGEX, é uma ferramenta de pesquisa e substituição de textos que pode ser bastante utilizada em diversos tipos de aplicações. Ela é uma composição de caracteres e símbolos chamados meta-caracteres, onde possuem a funcionalidade de pesquisar um padrão conforme as condições exigidas pela expressão. No caso das pesquisas, permitem escolher se a expressão a ser pesquisada estará em determinada posição da expressão, quais os caracteres que serão permitidos, quantas vezes a expressão deverá se repetir e uma inúmera série de opções que poderão ser planejadas para as pesquisas.

Elas destacam-se pela sua utilização em algumas situações distintas: Extração de dados: Acontece ao realizar uma exportação de dados para arquivos em formato texto; Pesquisa e substituição de textos: Utilizado para pesquisar informações específicas em arquivos ou para realizar modificações em lote de textos nos arquivos; Validação de dados: Servirá para a validação de campos de entrada quando a utilização de máscaras não são o suficiente para a validação dos dados.

Meta-caracteres

Na utilização de expressões regulares primeiramente deve ser aprendido o significado de cada palavra individualmente e, em seguida, utilizar a união dessas palavras em estruturas mais complexas e coerentes no contexto de uma determinada expressão, criando expressões regulares mais robustas usando o entendimento de cada meta-caractere separadamente.

Para sua utilização a linguagem de programação deverá dar suporte a expressões regulares, normalmente são utilizadas bibliotecas para expressões e o compilador é o responsável por analisar e interpretá-las para que o aplicativo forneça um resultado satisfatório quando receber algum dado a ser analisado. Além disso, cada meta-caractere é identificado por um mnemônico, um caractere que representará uma determinada regra de utilização.

Ponto - O sinal de ponto (.) é um meta-caractere utilizado em expressões, quando não se sabe exatamente qual o caractere que se quer pesquisar, aceitando qualquer caractere existente e obrigatoriamente deverá ser indicado um caractere, caso contrário, a expressão regular não será válida.

Asterisco - O asterisco (*) é um quantificador do tipo que aceitará nenhum. Os quantificadores indicam o número de repetições permitidas para o meta-caractere imediatamente anterior, ou seja, informam a quantidade de repetições que ele poderá conter.

Mais - O sinal de mais (+) é um quantificador que possui comportamento semelhante ao asterisco. A única diferença é que o mais não é opcional, nesse caso, o caractere anterior deverá aceitar pelo menos uma ocorrência.

Opcional - O opcional (?) é um meta-caractere quantificador semelhante ao ponto, diferindo na questão em que o caractere anterior ao opcional não é obrigatório, podendo conter ou não uma ocorrência desse caractere. Além disso, é possível definir como "opcional" regras definidas por outros meta-caracteres.

Escape - O escape (\) é utilizado para pesquisar um caractere que é igual a um dos meta-caracteres utilizados pelas expressões regulares. Assim, caso queira pesquisar um caractere de interrogação (?), deverá ser utilizado o escape para informar a expressão regular que o caractere (?) será descrito na expressão regular e que não representará o meta-caractere opcional.

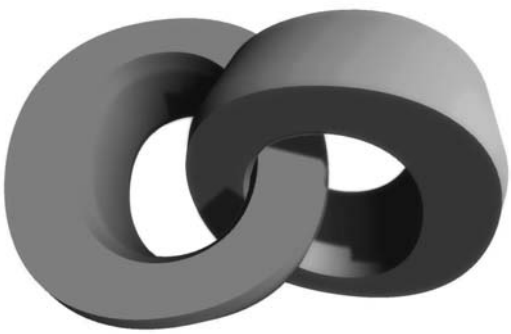
Circunflexo – O circunflexo (^) é uma espécie de apoio ou âncora que indica os caracteres que devem estar no começo de um texto a ser pesquisado.

Cifrão - O cifrão ($) é uma espécie de apoio semelhante ao circunflexo. A diferença é que os caracteres deverão estar no final do texto a ser pesquisado.

Ou - O meta-caractere ou (|) representa a possibilidade de se criar expressões com mais de uma alternativa de condição possível.

Grupo - O grupo (()) possibilita agrupar conjunto de caracteres e meta-caracteres em blocos, podendo assim ampliar as funcionalidades de determinados meta-caracteres e realizar agrupamentos.

Chaves - O meta-caractere chave ({m,n}) é um quantificador que permite especificar o número de repetições exato que um texto deverá se repetir para que a expressão regular seja verdadeira.



Lista - A lista ([]) determina um conjunto de caracteres que deverão estar presentes no texto a ser pesquisado. Na lista é possível realizar pesquisas utilizando o caractere hífen para separar a faixa de caracteres que serão pesquisados.

Outra possibilidade é utilizar determinadas palavras especiais, também chamadas de reservadas, que poderão definir um conjunto de caracteres padronizados. A essas palavras reservadas utiliza-se o nome de classes de caracteres POSIX.

A grande vantagem desse recurso é a transparência da expressão regular e a finalidade dessas classes poderem ser utilizadas em sistemas distintos que necessitem realizar pesquisas ou extração de conjunto de palavras escritas em outras línguas, pois, possibilitam utilizar caracteres especiais e acentuações.

A **Tabela 1** demonstra algumas expressões regulares utilizando classes de caracteres POSIX.

Lista Negada - A lista negada ([^]) possui comportamento igual ao da lista, mas indicará os conjuntos de caracteres que não deverão pertencer à expressão.

Para um melhor entendimento, vamos analisar a **Tabela 2**. Ela é um resumo que facilita o entendimento e indica os principais meta-caracteres utilizados e algumas de suas funcionalidades básicas, com exemplos válidos.

O uso das expressões regulares apresenta benefícios para a integração em diversas ferramentas e linguagens de programação, entre elas VB.NET, C#, Delphi Prism, Delphi XE, em aplicativos Linux/Unix como "vi" e no comando "grep". É uma linguagem dentro de uma linguagem permitindo utilizar seus recursos para benefício próprio e aprimoramento das expressões desenvolvidas, além de ser um ótimo modo de realizar pesquisas. Caso a palavra a ser pesquisada seja simples, a expressão também assim será. Vale lembrar que as palavras possuem diferenciação de maiúsculas e minúsculas retornando resultados diferentes. As expressões poderão conter intervalos, sequências, tipos distintos, validações, diferenciação de caracteres maiúsculos e minúsculos, quantificadores, enfim, um grande número de possibilidades para obter ou validar um determinado texto possibilitando realizar um grande número de combinações, tornando-se uma ferramenta poderosa para análise de dados com um padrão que possa ser identificado.

Nota do DevMan

POSIX (Portable Operating System Interface) é um conjunto de normas definidas pelo IEEE, que possuem o objetivo de garantir a portabilidade do código-fonte de um aplicativo a partir de um sistema operacional que atenda os requisitos das normas POSIX para outro sistema POSIX assim, as regras atuam como uma interface entre sistemas operacionais distintos facilitando a comunicação entre eles.

Delphi XE e Expressões Regulares

Uma das novidades no Delphi XE é a inclusão do suporte nativo a expressões regulares na sua RTL. Para fazer uso de uma expressão regular, basta adicionar a unit RegularExpressions na cláusula uses e utilizar o tipo TRegEx, que contém métodos estáticos para a utilização em expressões. O núcleo do mecanismo das expressões regulares em Delphi é o PCRE (Perl Compatible Regular Expression). A biblioteca apresenta um conjunto de funções que implementam a correspondência de padrão de expressões regulares usando a mesma sintaxe e semântica do Perl 5. PCRE tem sua própria API nativa, bem como um conjunto de funções que fornecem suporte ao padrão POSIX, facilitando o intercâmbio com outros sistemas e linguagens.

Vejamos alguns tipos usados no Delphi XE para suporte a expressões regulares:

TRegEx - É o tipo usado para o tratamento de expressões regulares. Executa a chamada às funções de TPerlRegEx que utilizam a interface das bibliotecas de PCRE. Suas principais funções são:

- IsMatch: utilizada para realizar comparações de um texto com uma expressão regular, onde o resultado é um valor booleano indicando se é um texto válido;
- Match: semelhante a IsMatch, mas retornando a primeira ocorrência válida como um record do tipo TMatch);
- Escape: substitui alguns caracteres especiais com seus códigos de escape;
- Matches: retorna todas as ocorrências válidas para a sequência de entrada, o resultado é armazenado em um tipo TMatch Collection;
- Replace: substitui todas as ocorrências de um texto utilizando determinado critério de substituição;
- Split: quebra um texto de entrada de acordo com uma regra de validação da expressão.

TMatch - É um record único que contém os resultados de uma validação de uma expressão regular. TMatch é instanciado pelo método de RegularExpressions.TRegEx. Match e seu valor representa o primeiro de uma sequência padrão utilizada. Se um padrão válido for encontrado, a propriedade Value conterá a substring de resultado, a propriedade Index a sua posição inicial e a propriedade Length o comprimento da substring encontrada. Como forma de agrupamento utiliza um TGroup, capaz de representar o resultado individual da pesquisa realizada.

POSIX	Expressão Regular	Significado
[:upper:]	[A-Z]	Letras maiúsculas
[:lower:]	[a-z]	Letras minúsculas
[:alpha:]	[A-Za-z]	Maiúsculas ou minúsculas
[:alnum:]	[A-Za-z0-9]	Letras maiúsculas ou minúsculas e números
[:digit:]	[0-9]	Somente números
[:xdigit:]	[0-9A-Fa-f]	Números em formato hexadecimal
[:punct:]	[.,!?:...]	Sinais de pontuação
[:blank:]	[\t]	Espaço e TAB
[:space:]	[\t\n\r\f\v]	Caracteres brancos

Tabela 1. Descrição de expressões regulares utilizando POSIX

Meta-caractere	Descrição	Exemplo	Expressão válida
.	Corresponde a qualquer caractere, exceto \n (quebra de linha).	a.c 1.0	abc a1c atc 1a0 110 1r0
[]	Combina qualquer caractere contido em uma faixa de valores determinados.	[abcd] [^abcd] [0-9] [a-z]	abcd cccc xyzl 0 1 2 3 4 r o d
\w	Indica qualquer caractere de palavra.	\w	a b c d eee
\W	Representa qualquer caractere que não seja palavra.	\W	\ & * % @
\s	Corresponde a qualquer caractere de espaço em branco.	\s	[espaço] [espaço]123
\S	Apresenta qualquer caractere não-espaço em branco.	\S	a b c d eee
\d	Indica qualquer dígito decimal.	\d	1 5 6 10
\D	Corresponde a qualquer caractere que não seja um dígito.	\D	a b c d eee
*	Indica zero ou mais ocorrências.	a*1 5*6	a1 1 aa1 55556 6
+	Representa uma ou mais ocorrências.	a+1	a1 aaaa1
?	Apresenta zero ou uma ocorrência, ou seja, torna o caractere anterior opcional.	a?1 5?6	a1 1 56 6
{n}	Indica exatamente n ocorrências.	\d{1} [a-z]{2}	1 4 5 6 aa ab bc
{n,}	Corresponde a pelo menos n ocorrências.	\d{1,} [a-z]{2,}	1 22 2345 aa abc aaaa
{n, m}	Representa pelo menos n, mas não mais que m ocorrências.	[a-z]{1,1}	a b c d
^	Corresponde ao início de uma linha.	^[a-z]	a b c d123
$	Representa o final da linha.	$[a-z]	123a a b c
\b	Indica o início da palavra.	\bdel	delphi delegado
\	Inibe a funcionalidade de um meta-caractere.	\. \?	. ?
|	Alternativo (ou).	[0-1]|[1-2]	1 01 12 2 21

Tabela 2. Descrição de metacaracteres

TMatchCollection - Contém um conjunto de instâncias de TMatch, onde cada ocorrência representa uma sequência válida para a expressão regular. Possui um contador de índices representado por Count, um array TMatch indicado por Item[index: Integer] para representar cada ocorrência válida da expressão e uma função GetEnumerator para percorrer as suas instâncias.

TGroup - Um record que contém os resultados de uma única validação de uma expressão regular. Por se tratar de um agrupamento de um determinado resultado, deverá ser utilizado em conjunto com TMatch, pois dependerá da validação inicial da expressão regular. Dessa forma, um TGroup poderá ser indexado de maneira que se possa utilizar seu array de itens para manter constante os valores de suas propriedades individuais. Apresenta as funções GetLength (informando o seu tamanho), GetIndex (representando o índice) e GetValue (para visualizar a string de resultado).

TGroupCollection - Representa uma coleção de itens de TGroup. Semelhante ao anterior, seu funcionamento e utilização depende de uma validação de expressão utilizando o acesso a chamada de RegularExpressions.TMatch.Groups. Possui a função GetEnumerator (para indicar os grupos da coleção) e uma propriedade Item, onde este array retorna um TGroup resultante da validação da expressão regular utilizada.

Exemplos no Delphi XE

Na prática, no Delphi XE, a **Listagem 1** indica o código de uma função para validar endereço IP utilizando expressões regulares.

O código da **Listagem 1** mostra um exemplo usando diretamente TRegEx, essa forma é permitida, pois trata-se de um teste simples de validação de uma expressão sem a necessidade de se criar uma variável desse tipo.

Realizando uma análise na expressão regular para validação de um endereço IP é possível constatar:

\b (que corresponde ao início da palavra);

25[0-5] (composto de 25 + numeral de 0 a 5) ou;

2[0-4][0-9] (2 mais 0 a 4 mais 0 a 9) ou;

[01]?[0-9][0-9]? (0 a 1 o meta-caractere opcional, indicando a não obrigatoriedade de [01] mais numeral de 0 a 9 e outro numeral de 0 a 9).

Dessa forma endereços iniciais poderão ser configurados como válidos: 255, 204, 249, 192, 189, 100, 099, 90 etc. Além disso,

o escape \. indicará que conterá um ponto (.) após o numeral acima referenciado e, para as demais sequências, a lógica a ser seguida será semelhante.

A **Listagem 2** representa a utilização de uma pesquisa mais robusta retornando os dados encontrados como mensagem na tela para o usuário.

A definição de expressaoregular corresponde ao tipo TRegEx contendo uma expressão regular descrita por [0-9], ou seja, um numeral.

Match do tipo TMatch executa a pesquisa em uma String e o resultado é armazenado na variável grupo do tipo TGroup realizando uma estrutura de repetição e continuando a pesquisa através de NextMatch. Deve ser observado que a mensagem indica o valor número de retorno, representado por uma String, para o usuário através de ShowMessage.

Listagem 1. Código para validar um endereço IP utilizando expressões regulares

```
function ValidaEnderecoIP(ip: string): boolean;
var
  ipRegExp : String;
begin
   try
     ipRegExp := '\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-  4][0-9]|[01]?[0-9][0-9]?)\.
     (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b';
     if TRegEx.IsMatch(ip, ipRegExp) then
       result := true
     else
       result := false;
     except
       result := false;
   end;
end;
```

Listagem 2. Definição de uma pesquisa avançada

```
procedure TForm2.Pesquisa(palavra : string);
var
  expressaoregular: TRegEx;
  match: TMatch;
  grupo: TGroup;
begin
  expressaoregular := TRegEx.Create('[0-9]',[roIgnoreCase,roMultiline]);
  match := expressaoregular.Match(palavra);
  while match.Success do
  begin
    grupo := match.Groups.Item[0];
    ShowMessage('Valor:' + grupo.Value);
    match := match.NextMatch;
  end;
end;
```

Listagem 3. Definição de script SQL para banco de dados utilizando expressões regulares

```
create table aluno (
  numero numeric(3) not null,
  nome varchar(25) not null,
  fone varchar(14)
  check (fone similar to '\([0-9]{2}\) [0-9]{4}\-[0-9]{4}' escape '\')
);

insert into aluno
  values ('000', 'Rodrigo Araujo', '(51) 5555-1234');

insert into aluno
  values ('000', 'Devmedia', '(11) 1234-1234');

insert into aluno
  values ('100', 'teste de aluno', '(55) 5555-123'); --erro violação de check constraint

select * from aluno
  where fone not similar to '\([0-9]{2}\) 5555\-%' escape '\';
```

Expressões Regulares em Banco de Dados

As expressões regulares, além de serem utilizadas no desenvolvimento de aplicativos com o Delphi XE, podem ser utilizadas em banco de dados em instruções SQL. Como exemplo, um banco de dados que possui suporte a estas funcionalidades é o Oracle em sua versão 10G, ou superior, possuindo as seguintes funções de busca baseadas no uso de expressões regulares: REGEXP_LIKE(), REGEXP_REPLACE(), REGEXP_SUBSTR().

Outro exemplo é o Firebird, em sua versão 2.5, que possui um novo predicado SIMILAR TO introduzido para dar suporte a expressões regulares. A função do predicado é verificar se uma determinada expressão padrão SQL corresponde a um argumento válido em qualquer contexto que poderá aceitar expressões booleanas, como cláusulas WHERE, as restrições CHECK e testes com PSQL (IF). A **Listagem 3** demonstra um script de código SQL no Firebird.

A instrução create table possui uma expressão regular de validação de telefone no campo fone, representado por check e pela expressão '\([0-9]{2}\) [0-9]{4}\-[0-9]{4}'. Os registros inseridos na tabela aluno são analisados verificando a constraint do campo fone. Ocorrerá um erro ao inserir o registro 'teste de aluno', pois o telefone não satisfaz as condições da expressão regular (ao invés de quatro dígitos finais, possui três). Ao realizar o select, contendo uma expressão regular representada pela negação da condição exposta, apenas o registro de aluno 'Devmedia' satisfará a expressão e será apresentado como resultado.

Conclusão

As expressões regulares são de fácil utilização e normalmente economizam trabalho relacionado à pesquisa e substituição de textos. Elas proporcionam uma boa eficiência ao aplicativo em que for

utilizado, mas suas definições de regras devem ser feitas de forma clara para a validação de informações, levando em conta o conhecimento obtido no aprendizado de meta-caracteres.

Uma utilização interessante é na coleta de determinado tipo de informações de arquivos, como por exemplo, a possibilidade de coletar informações diretamente de sites que informem cotações da bolsa de valores, indicando uma expressão regular para o nome de uma empresa e outra expressão para o valor da cotação desta empresa.

Além disso, elas podem auxiliar no controle do fluxo de entrada de dados de uma aplicação, gerenciando as regras de validação das informações de entrada, diminuindo, dessa forma, os erros de digitação do usuário de um sistema, ou seja, validando as informações de todo o aplicativo.

Rodrigo Araujo
rodrigo.arsi@gmail.com
Bacharel em Ciência da Computação pela UFSM – Santa Maria/RS trabalha com Análise e Desenvolvimento de Software há mais de 10 anos. Desde 2003, utiliza tecnologias .NET.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Regular Expressions - XE API Documentation
http://docwiki.embarcadero.com/VCL/en/RegularExpressions

Expressão Regular
http://pt.wikipedia.org/wiki/Express%C3%A3o_regular

VCL and RTL Changes for XE - RAD Studio XE
http://docwiki.embarcadero.com/RADStudio/XE/en/VCL_and_RTL_Changes_for_XE

PCRE - Perl Compatible Regular Expressions
http://www.pcre.org/

Cursos Online

Assinatura .net magazine PLUS
Mais conteúdo .NET por muito menos!

A Revista **.net Magazine** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/netmagazine

- **Crie uma loja Virtual completa**
- **Construindo relatórios com Crystal Reports e Visual Studio 2005**
- **Criando uma aplicação Web Completa**
- **Criando uma aplicação client/server no Visual Studio 2005**
- **Aprenda a criar um blog com ASP.NET**
- **Curso de C#**

A sua melhor opção de aprendizagem!
Assine a **.net Magazine** e
Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

DataSnap XE, Generics
Transfira e persista objetos em

No que tange a área de desenvolvimento, a criação dos padrões de projetos trouxe um enorme benefício uma vez que apresentam soluções para problemas recorrentes no desenvolvimento de projetos orientados a objetos. Com isso, ao iniciar um projeto já sabemos o caminho a seguir para evitar determinado problema. A orientação a objetos evoluiu, ou melhor, as linguagens orientadas a objetos começaram a incorporar novos recursos, recursos estes que vieram trazer enorme benefício ao desenvolvimento de software orientado a objetos. As linguagens e ferramentas também evoluíram incorporando em seus IDEs novas Features, Wizards e utilitários, sempre com o foco na produtividade.

Quando falamos de Delphi também não ficamos atrás. A linguagem Delphi evoluiu nos últimos anos incorporando alguns destes recursos como Generics, Anonymous Methods, nova RTTI, DBX4 e muito mais. Este ano tivemos o lançamento do Delphi XE que conta com uma série de ferramentas incorporadas ao IDE como integração com Subversion, CodeSite, FinalBuilder, entre outros.

Um capítulo à parte quando se trata de Delphi está por conta da arquitetura DataSnap. Não podemos negar que o desenvolvimento em camadas sempre foi um diferencial no Delphi. Aplicações em camadas sempre foi uma boa opção para projetos grandes que demandam escalabilidade e facilidade de manutenção. Com este tipo de aplicação podemos concentrar nossas regras de negócio no servidor criando assim aplicações clientes mais enxutas. A distribuição da aplicação também é facilitada, uma vez que não há necessidade de instalação de drivers para acesso a dados ou qualquer outra tecnologia de conexão a banco de dados. Nas últimas versões do Delphi a arquitetura DataSnap evoluiu e muito, permitindo que os desenvolvedores Delphi possam lançar mão do que há de mais recente no mundo do desenvolvimento, como Cloud Computing, RIA, aplicações RESTFull e muito mais.

Agora imagine o quão interessante seria unir toda a produtividade do Delphi em sua arquitetura DataSnap com padrões e recursos avançados da orientação a objetos. Atente para o fato de todos os padrões de desenvolvimento, framework ágil, metodologia ou filosofia de desenvolvimento não está vinculada a linguagem, ou seja, não existe restrição quanto à linguagem e/ou ferramenta utilizada para você desenvolver uma aplicação baseado em RIA por exemplo. Muito menos alguma exigência para a aplicação do padrão DAO (veremos mais adiante) a não ser que a linguagem que você for trabalhar seja orientada a

Resumo DevMan

De que trata o artigo:

O artigo trata do desenvolvimento de aplicação DataSnap utilizando a nova arquitetura DataSnap XE, lançando mão de recursos avançados da orientação a objetos, como a reflexão computacional, padrões de projeto como DAO além de Generics, um dos recursos mais interessantes disponíveis na linguagem Delphi.

Para que serve:

O desenvolvimento em camadas serve para criarmos aplicações escaláveis, de fácil manutenção e distribuição, uma vez que as regras de negócios ficam encapsuladas no servidor. Desenvolver uma aplicação em camadas com recursos de OO como Generics e RTTI aumenta a produtividade da equipe permitindo um maior reaproveitamento de códigos e como consequência uma redução de prazo e custo.

Em que situação o tema é útil:

Utilizar conceitos avançados de desenvolvimento orientado a objetos é útil quando se quer aumentar o nível de abstração do projeto e como consequência tornar o código mais coeso e menos acoplado. Assim a aplicação fica muita mais limpa, fácil de manter e muito flexível quando há a necessidade de implementar novos recursos evitando o retrabalho e efeitos colaterais.

DataSnap XE, Generics, RTTI e DAO:

Nos últimos tempos assuntos como padrões de projetos, metodologias ágeis, boas práticas entre outros estão sendo discutidos como nunca. Com o avanço da tecnologia, prazos cada vez menores e clientes cada vez mais exigentes, a busca por métodos e procedimentos que aumentem a produtividade se torna uma obrigação. No desenvolvimento orientado a objetos existem alguns paradigmas que proporcionam maior produtividade como o uso da herança e polimorfismo, mas somente estes paradigmas de forma isolada não trazem o benefício esperado no atual cenário tecnológico em que estamos. As equipes estão cada vez mais heterogêneas, os projetos cada vez maiores, as opções de linguagens e frameworks são bem vastas e para manter tudo isso muito bem alinhado é preciso uma gestão eficiente. Mas isso é mais assunto para gerência de projeto. Neste artigo vamos observar como estes avanços influenciam a vida do desenvolvedor e como podemos aplicá-los em nossos projetos desenvolvidos em Delphi.

, RTTI e DAO aplicações multicamadas

Rodrigo Carreiro Mourão

objetos. O mesmo vale para os padrões de projeto. Como o Delphi é uma linguagem 100% orientada a objetos fica fácil entender que todos estes padrões podem ser aplicados em soluções desenvolvidas em Delphi e esta é a proposta deste artigo.

Vamos ver como aplicar alguns dos padrões de desenvolvimento e ainda, lançar mão de recursos avançados da orientação a objetos como Generics, trabalhar com reflexão computacional através da RTTI do Delphi, tudo isso em uma aplicação em camadas utilizando a nova arquitetura DataSnap.

Generics

Generics é um dos recursos mais interessantes em Orientação a Objetos em minha opinião, pois flexibiliza muito a escrita do código. Como o próprio nome sugere este recurso nos permite trabalhar com uma estrutura genérica, o que faz toda a diferença no caso do Delphi que é uma linguagem fortemente tipada. Quando falamos que uma linguagem é fortemente tipada significa que as estruturas definidas nesta linguagem obrigatoriamente têm que ter um tipo de dado no momento de sua declaração e este tipo de dado não poderá mudar, de uma forma geral seria dizer que o que nasce inteiro, vive inteiro e morre inteiro.

Ocorre que um dos pilares da Orientação a Objetos é o polimorfismo e para que possamos implementar o polimorfismo temos que lançar mão da herança, onde geralmente temos uma classe básica da qual descendem os demais objetos. Com isso definimos uma variável deste tipo básico e instanciamos um objeto de um tipo descendente, até aí tudo perfeito. Mas quando precisamos acessar um método ou propriedade exclusiva do objeto descendente somos obrigados a fazer um Type Casting, e esta tarefa onera e muito o processamento além de permitir que erros ocorram.

Através dos Generics pode-se definir uma estrutura genérica customizada, ou melhor, tipada de acordo com a necessidade. O objeto, parâmetro ou variável não tem seu tipo definido no momento da declaração e sim no momento do uso. Um caso clássico para quem trabalha com orientação a objetos são as listas de objetos. É muito comum utilizar a classe TList para trabalhar com listas de objetos, porém a TList pode conter qualquer tipo de objeto e quando se acessa um item da lista recebe-se um TObject, o que nos obriga a realizar o Type Casting. Sem falar que não há garantias de que em uma lista de clientes, por exemplo, não será inserido um botão, pois não há validação em tempo de projeto. Uma solução seria criar uma lista exclusiva de clientes herdando de TList, mas isto pode se tornar um problema se tivermos que fazer isso para cada tipo de objeto em uma aplicação, tornando-se inviável.

Com Generics isso é resolvido de uma forma extremamente simples, utilizando

Nota do DevMan

Anonymous Methods é um recurso que foi adicionado no Delphi 2009 e é conhecido também como "referências a métodos", pois sua declaração é feita com a sintaxe "reference to function/procedure". Em resumo criar um método anônimo é criar um tipo que faz referência a um procedure ou function com uma assinatura específica, por exemplo:

```
TCalculo = reference to function(const N1, N2: double): double;

procedure ExecutaCalculo(Calculo: TCalculo);

ExecutaCalculo(function(const N1, N2: double): double
                 begin
                 result := N1 + N2;
                end);
```

Nota do DevMan

O Delphi XE trouxe como principal atrativo a integração do IDE com diversas ferramentas utilitárias há muito utilizadas pela comunidade, são elas:
Subversion - também conhecido por SVN, é um sistema de controle de versão desenhado especificamente para ser um substituto moderno do CVS, que se considera ter algumas limitações.
FinalBuilder - ferramenta para gerenciamento de compilações, que traz uma maneira fácil e prática de criar e rodar scripts de compilações. Com uma interface amigável para o usuário esse software permite realizar a compilação de um ou vários projetos.
CodeSite - é como um super gerador de arquivos de log, pois permite enviar dados dos erros para o log, inclusive tipos personalizados. O log pode então ser visualizado localmente, no viewer ou gravado num arquivo. As mensagens podem ser de erros, avisos, informações, sendo mostradas com ícones diferentes, podendo ser filtradas, para esconder mensagens desnecessárias.
AQTime - ferramenta para analisar a performance do código-fonte, facilitando o descobrimento dos gargalos, permitindo otimizações mais precisas mostrando onde realmente está o problema. AQTime é ao ferramenta de Profile líder há anos entre os desenvolvedores Delphi.

Nota do DevMan

Desenvolvimento ágil de software ou método ágil é um conjunto de metodologias. Providencia uma estrutura conceitual para reger projetos de Engenharia de Software. Entre as metodologias mais conhecidas podemos citar o XP, Scrum, FDD, DSDM etc.

Nota do DevMan

RIA - Rich Internet Application - são Aplicações Web que têm características e funcionalidades de softwares tradicionais do tipo Desktop. RIA típicos transferem todo o processamento da interface para o navegador da internet, porém, mantêm a maior parte dos dados (como por exemplo, o estado do programa, dados do banco) no servidor.

Nota do DevMan

O conceito de computação em nuvem (em inglês, Cloud Computing) refere-se à utilização da memória e das capacidades de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet, seguindo o princípio da computação em grade (grid). O acesso a programas, serviços e arquivos é remoto, através da Internet - daí a alusão à nuvem. O uso desse modelo (ambiente) é mais viável do que o uso de unidades físicas.

Nota do DevMan

Type Casting ou Type Convertion é um recurso que muitas linguagens de programação possuem e que fica mais evidente seu uso nas linguagens ditas fortemente tipadas. A função do Type Casting, como o nome sugere, é realizar a conversão de tipo de um determinado objeto. Em POO é muito comum este tipo de procedimento quando trabalhamos com objetos em estrutura hierárquica, onde um objeto de um determinado tipo é convertido para outro para que se possa acessar uma propriedade ou método específico.

Nota do DevMan

A Orientação a Objetos é baseada em 4 pilares a saber: Herança, Polimorfismo, Encapsulamento e Abstração.
Herança ou generalização é o mecanismo pelo qual uma classe pode estender outra classe, aproveitando seus métodos e propriedades.
Encapsulamento consiste na separação de aspectos internos e externos de um objeto e é utilizado amplamente para impedir o acesso direto ao estado de um objeto.
Abstração é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.
Polimorfismo consiste na capacidade que uma linguagem tem de fazer um mesmo método se comportar de maneira diferente de acordo com o objeto de onde foi chamado.

Nota do DevMan

Utilizar a classe TList tipada é muito simples, basta no momento da declaração da variável informar qual o tipo de objeto deseja guardar na lista. Observe:

```
var
  Lista: TList<TCliente>;
begin
  Lista := TList<TCliente>.Create;
  (...)
end;
```

a classe TList tipada, ou seja, no momento do uso você define qual tipo de dados a lista vai armazenar. Com isso o parâmetro solicitado no método Add será do tipo definido na criação da lista, assim como todos os demais métodos e propriedades que se referem ao tipo.

RTTI

Outro recurso muito importante na orientação a objetos é a reflexão computacional. Este recurso trata da capacidade que a linguagem tem de extrair informações de um objeto em tempo de execução, daí a sigla RTTI que significa Run Time Type Information ou Informação de tipo em tempo de execução. Um exemplo clássico de RTTI no Delphi que é muito citado para ilustrar o recurso é o Object Inspector. A função dele dentro do IDE é permitir o desenvolvedor alterar as propriedades de um dado componente, neste caso, o componente selecionado no momento. Não podemos esquecer que este componente é na verdade um objeto de um tipo específico, seja ele um TButton, TEdit, TTable etc. E para cada componente selecionado o Object Inspector mostra suas respectivas propriedades e eventos.

Note que as propriedades são montadas dinamicamente e isso só é possível devido a RTTI. Quando o objeto é selecionado, via RTTI o IDE extrai todas as propriedades publicadas do componente em questão para então permitir-nos alterá-las conforme a necessidade. E também, através de RTTI é que os valores das propriedades são alterados nos objetos que estão instanciados em memória. Com isso chega-se a conclusão que reflexão computacional nos permite extrair informações de um objeto e também alterar os valores de suas propriedades.

Até o lançamento do Delphi 2010 a RTTI era muito engessada e limitada, mas com a chegada desta nova versão a coisa mudou de figura. Novas classes foram introduzidas da VCL e RTL, o compilador passou a dar mais informações sobre os objetos, a RTTI ficou muito mais flexível e intuitiva e o melhor, muitas das limitações foram jogadas por terra. Isso quebrou uma barreira muito grande, uma vez que o Delphi é compilado em Delphi, pois com uma flexibilidade maior, novos recursos podem ser adicionados ao IDE. Sem falar que novos frameworks e novos componentes podem e já começaram a surgir.

Lançar mão de estruturas baseadas em RTTI significa adicionar aos projetos orientados a objetos uma flexibilidade muito grande. Utilizar RTTI é elevar o nível de reutilização de códigos, é permitir que uma mesma estrutura possa atender as mais diversas situações, sem falar que a manutenção se torna mais simples e rápida. Prova disso são os frameworks de mapeamento objeto relacional que existem no mercado, não só para Delphi, mas para outras linguagens que se utilizam de reflexão computacional para realizar o que chamamos de mapeamento objeto relacional, extraindo informações dos objetos em memória para gerar as instruções SQL e então, persisti-los em banco de dados.

Nota

Entende-se por algumas limitações da antiga estrutura RTTI do Delphi o fato de se dever declarar as propriedades como published, a necessidade das diretivas {$M+}, o fato de não ter como identificar os métodos e propriedades de um objeto e os métodos e propriedades que este mesmo objeto herdou de seu ancestral, entre outras.

Nota do DevMan

O Delphi 2010 introduziu uma nova RTTI com novas classes, tipos, métodos, units e funções. Como principal estrutura temos a presença do record TRttiContext que extrai as informações do tipo que está sendo mapeado, além de um tipo para cada estrutura utilizada nas classes que desenvolvemos, como por exemplo, TRttiMethod, TRttiProperty, TRttiType, TRttiParameters, TRttiAttributes etc.

Nota do DevMan

Mapeamento objeto relacional ou ORM, do inglês: Object Relational Mapping é uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada a objetos utilizando bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes. Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL, devendo apenas usar uma interface de programação simples que faz todo o trabalho de persistência.

DAO

DAO significa Data Access Object ou Objeto de Acesso a dados. É um padrão de desenvolvimento de softwares que trata da persistência de objetos e prega a separação das camadas de negócio e acesso a dados. De uma forma mais tradicional, para cada classe de negócio há uma classe de persistência correspondente que tem a função de encapsular toda a comunicação com o banco de dados para a realização do procedimento de CRUD. O padrão DAO está geralmente associado a aplicações J2EE, mas seu conceito pode ser aplicado a qualquer linguagem dita orientada a objetos.

A principal vantagem deste tipo de padrão é justamente a possibilidade de manter em estruturas separadas todos os comandos relativos à conexão com banco de dados, consultas SQL etc., permitindo assim que o código da aplicação fique mais coeso. Sem falar que como a parte de acesso a dados está encapsulada em uma camada separada, há a flexibilidade de alterar a tecnologia de acesso a dados ou mesmo o banco de dados a se persistir os objetos, sem que se tenha que alterar as classes de negócio e o funcionamento da aplicação.

O problema é que o número de classes aumenta consideravelmente no caso de se optar pelo padrão DAO para persistência dos dados, mas lançando mão de recursos como RTTI e Generics, pode-se criar um mecanismo dinâmico para geração de comando SQL sem que para isso tenha-se que criar um para cada classe de negócio. Esta é a ideia deste artigo, mostrar que existem alternativas e principalmente, possibilidades.

Aplicação Servidora

Os conceitos vistos até aqui fazem mais sentido quando são colocados em prática em uma aplicação DataSnap, e isso pode ser feito criando uma nova aplicação no Delphi XE. Para isso deve ser selecionado um modelo de DataSnap Server. A **Figura 1** exibe o Wizard que permite selecionar os protocolos de transporte, suporte a autenticação e inclusão de métodos de exemplo.

A interface do servidor de aplicação não possui nenhum recurso extraordinário até porque o foco está nos serviços, ou melhor, nos métodos que o servidor irá disponibilizar para seus clientes. A **Figura 2** mostra uma sugestão de interface apenas com um pequeno texto e dois botões, um para iniciar o servidor e outro para interrompê-lo. Para os botões de Start e Stop sugere-se os códigos da **Listagem 1**.

Desde a versão 2010 do Delphi foi adicionado ao framework DataSnap um novo e forte aliado, chamado JSON. Os dados podem ser trafegados entre o cliente e o servidor em notação JSON, facilitando assim o desenvolvimento. Desta forma, um objeto de negócio na aplicação pode ser convertido para notação JSON, enviado ao servidor de aplicação para que possa ser novamente convertido para notação tradicional e persistido na base de dados.

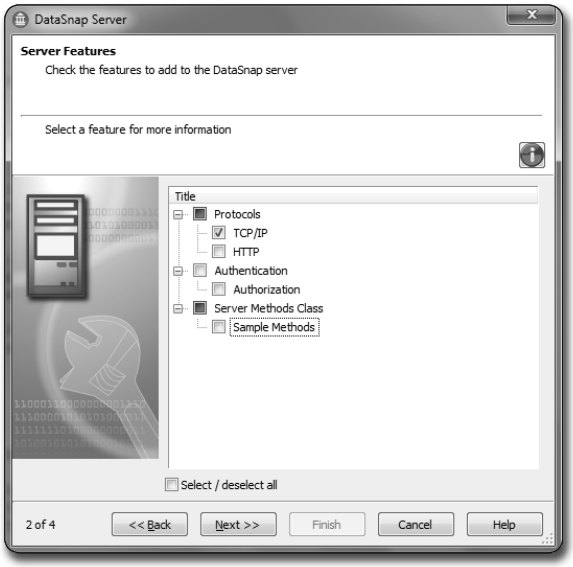


Figura 1. Criando o servidor de aplicação

Figura 2. Interface do servidor

A VCL traz para algumas de suas units, classes e métodos específicos para se trabalhar com objetos e dados em notação JSON. Os dois métodos principais são o Marshal, que tem por objetivo converter um objeto tradicional em um objeto JSON e o método unMarshal que tem função oposta, ou seja, converter um objeto JSON em um objeto tradicional.

O JSON (Javascript Object Notation) é uma espécie de padronização para transporte de informações entre ambientes heterogêneos e que tem como linguagem o Javascript. Padrão, transporte, ambiente heterogêneo, tudo isso remete a XML, então eis a questão: Por que JSON e não XML? O principal motivo seria pelo fato da notação JSON ser extremamente mais simples do que a do XML. Não existem tags diversas nem regras complexas para escrita e muito menos mecanismos complexos de parser.

Classes de negócio

Em uma aplicação real temos muitas classes de negócio com associações com outras classes e muitas outras regras de negócio. Torna-se inviável criar um método para converter cada uma das classes com que se pode trabalhar, sem falar que novas classes podem surgir. É neste momento que o uso do recurso de Generics

Listagem 1. Código dos botões

```
if not SC.DSServer.Started then
begin
  SC.DSServer.Start;
  BtnStart.Enabled := False;
  btnStop.Enabled  := True;
end;

if SC.DSServer.Started then
begin
  SC.DSServer.Stop;
  BtnStart.Enabled := True;
  btnStop.Enabled  := False;
end;
```

Nota do DevMan

CRUD é o acrônimo de Create, Retrieve, Update e Delete que são as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface para usuários para inserção, consulta, atualização e exclusão de dados.

Nota do DevMan

Como o nome indica, as constraints em Generics nada mais são do que restrições sobre os tipos reais que podem ser usados para parametrizar um tipo genérico. Ou seja, quando em tempo de execução definir é definido o tipo real com que se vai trabalhar, este tipo deve satisfazer a condição definida. Atualmente pode-se definir 4 tipos de restrições com suas variações:

Class - Define que o tipo real a ser utilizado deve obrigatoriamente ser um objeto descendente de TObject, exemplo de definição: myfunction<T: class>(Obj: T). Caso queira especificar que este objeto deve herdar de uma classe mais específica basta definir: myfunction<T:TCustomEdit>(Obj:T). Neste caso o objeto real passado deve ser obrigatoriamente descendente da classe TCustomEdit.

Interface – Define que o tipo real a ser utilizado deve obrigatoriamente descender de uma determinada interface, ou ser um objeto que implemente esta interface, por exemplo: myfunction <T: IInterfaceList>(Obj:T). Neste caso o objeto passado deve ser do tipo desta interface, descendente desta interface ou implementar esta interface.

Record - Define que o tipo real a ser utilizado deve ser obrigatoriamente uma estrutura de valor e não de referência, podendo ser um valor ordinário, ponto flutuante ou um Record, por exemplo: myfunction<T: record>(Obj:T).

Constructor - Define que o tipo real a ser utilizado deve possuir obrigatoriamente um construtor declarado que não possua parâmetros. Isso é muito comum quando internamente na função há a necessidade de se instanciar este tipo genérico. Assim há a certeza que pode-se invocar o método T.create sem que hajam erros em tempo de execução. Um exemplo de definição seria: myfunction <T: constructor>(Obj:T).

se encaixa perfeitamente. Pode-se ter um método que converta um tipo genérico para um objeto JSON, tipo este definido em tempo de execução e da mesma forma um segundo método que receba um objeto JSON e o converta em um objeto de tipo genérico, que também será definido em tempo de execução.

A **Listagem 2** mostra um exemplo de como isso pode ser implementado. Primeiro é definida uma classe chamada de TBase contendo dois métodos de classe. Cada método define um tipo genérico, representado pela letra T. Este tipo genérico possui a constraint class. Este também especifica o tipo dos parâmetros bem como de retorno no caso do método JSONtoObject, assim, se em tempo de execução definir o tipo como um TButton logo o parâmetro será do tipo TButton e o retorno da função JSONtoObject será do tipo correspondente.

Para a conversão de um objeto JSON para um objeto tradicional, é usada a classe TJSONUnMarshal, que através do método unMarshal realiza todo o trabalho pesado. Ainda é possível notar um Type Casting do objeto retornado pela função para o objeto do tipo T, ou seja, o tipo genérico que será definido em tempo de execução. Já o processo para a conversão de um objeto tradicional para notação JSON faz uso da classe TJSONMarshal que invoca o método Marshal, onde da mesma forma que no processo anterior, faz todo o trabalho de conversão. Caso o objeto passado não esteja devidamente instanciado a função retorna um objeto JSON nulo evitando assim erros de Access Violation.

Métodos de classe nada mais são do que métodos que pertencem à classe e não ao objeto. Assim, não se faz necessária a criação do objeto para invocar o método, podendo invocá-lo diretamente da classe. Suponha que a classe TMyClass possua um método de classe DoFoo, para invocá-lo basta fazer TMyClass.DoFoo, não sendo necessário instanciar o objeto. Para se definir um método como sendo da classe, basta precedê-lo da palavra class, por exemplo, class procedure e class function.

Com a classe Base implementada tudo fica muito simples, toda e qualquer classe que herde desta terá a capacidade de converter qualquer tipo em um objeto JSON e vice e versa. Um detalhe importante a ser citado é que uma classe qualquer como, por exemplo, TCliente, pode ter seu método ObjectToJSON invocado para converter um objeto do tipo TFornecedor. O tipo não está restrito à classe que invoca o método e sim ao tipo genérico que é definido no momento de utilização do método.

Na **Listagem 3** é possível ver a declaração da classe TContato utilizada para demonstrar a transferência de objetos entre a aplicação cliente e a aplicação servidora.

Listagem 2. Classe base com métodos de conversão

```
uses DBXJSON, DBXJSONReflect;

TBase = class
public
  class function ObjectToJSON<T: class>(aObject: T): TJSONValue;
  class function JSONToObject<T: class>(aJSON: TJSONValue): T;
end;

class function TBase.JSONToObject<T>(aJSON: TJSONValue): T;
var
UnMarshal: TJSONUnMarshal;
begin
if aJSON is TJSONNull then
    Exit(nil);
  UnMarshal := TJSONUnMarshal.Create;
  Result := T(UnMarshal.Unmarshal(aJSON));
  UnMarshal.Free;
end;

class function TBase.ObjectToJSON<T>(aObject: T): TJSONValue;
var
Marshal: TJSONMarshal;
begin
if Assigned(aObject) then
begin
  Marshal := TJSONMarshal.Create(TJSONConverter.Create);
  Result := Marshal.Marshal(aObject);
  Marshal.Free;
end
else
  Exit(TJSONNull.Create);
end;
```

Classe de conexão

Como citado no início deste artigo uma dos padrões de desenvolvimento apresentado aqui é o DAO, com isso é fato que se fará necessário ter uma conexão com o banco de dados para que os objetos possam ser persistidos. Sendo assim, uma das maneiras que temos para controlar o acesso a dados em uma aplicação orientada a objetos é ter uma classe que gerencia esta conexão dentro da aplicação. Como geralmente em uma aplicação é utilizado apenas um banco de dados, nada melhor do que aplicar o padrão de projeto Singleton.

A classe de conexão utilizada para acesso ao banco de dados está descrita na **Listagem 4** e se faz necessário ressaltar alguns pontos. O primeiro deles é o fato do construtor da classe gerar uma exceção. Isso é necessário, pois como o objeto implementa o padrão de projeto Singleton, logo o canal para criação deste deve ser apenas o método GetInstance. A função do método GetInstance é verificar, através da variável de classe FInstance, se o objeto já está instanciado e em caso afirmativo este objeto é retornado, caso contrário o primeiro objeto é criado e armazenado nesta mesma variável. Outro detalhe está

no construtor privado, onde a conexão com o banco de dados é realizada utilizando as classes da versão 4 do DBX framework. Através da classe TDBXConnectionFactory foi solicitado ao framework uma nova conexão com o banco de dados cujo os parâmetros de conexão são:

• DBContato - Nome de uma conexão com o banco de dados MySql registrada no IDE do Delphi. Neste banco de dados há uma tabela chamada TBL_CONTATO cujo script pode ser visto na **Listagem 5**;

• root – Usuário utilizado para acessar a base de dados;

• root – Senha utilizada para acessar a base de dados.

O objeto de conexão com o banco de dados é então armazenado no Field FConnection, e em seguida um objeto do tipo TDBXCommand é criado. Este será responsável por executar os comandos SQL necessários no banco de dados. Por fim o objeto DBXCommand é exposto através da propriedade Command, permitindo assim que comandos SQL possam ser enviados a classe de conexão e então executados no banco de dados.

Atributos customizados

Um dos novos recursos disponíveis no Delphi é a possibilidade de utilizar atributos customizados. Aqui são definidos dois atributos customizados, um para definir o nome da tabela em que o objeto deverá ser persistido e outro para definir quais das propriedades serão utilizadas como chave para que ao ser mapeadas as instruções de update e delete possamos saber exatamente qual objeto remover do banco de dados.

O Atributo TableName está exposto na **Listagem 6** enquanto o atributo KeyField é descrito na **Listagem 7** e ambos possuem apenas uma propriedade a saber, a propriedade Name, que é utilizada para que se possa acessar o conteúdo a ser utilizado no mapeamento objeto relacional.

Com os atributos já definidos podemos voltar a classe TContato e inserir os dois atributos que serão utilizados para realizar o mapeamento objeto relacional. Sobre a declaração da classe é adicionado o atributo TableName e nele é inserido o nome da tabela no banco de dados que conterá os dados da classe TContato. Já sobre a propriedade OID é inserido o atributo KeyField e como valor do atributo informamos o nome do campo chave primária na tabela tbl_contato. Na **Listagem 8** a seguir é possível ver como os atributos são declarados na classe TContato.

Classe DAO Genérica

Uma vez que as estruturas necessárias para o início do mapeamento objeto relacional já estão prontas é possível definir a classe que irá realizar este mapeamento (**Listagem 9**). Como foi mostrado anteriormente, o padrão DAO prega que para cada classe de negócio deve existir uma classe de mapeamento que seja responsável pela persistência. Neste artigo está sendo empregado o recurso de Generics para que possa ser criada uma estrutura que atenda a todas as classes de uma forma geral. Há nesta estrutura um método para cada tipo de operação a ser realizada no banco de dados, além de dois métodos adicionais: GetTableName que tem a função de retornar o nome da tabela que será utilizada para persistir o objeto que esta sendo mapeado no momento, e GetAll que tem a função de retornar um objeto do tipo TDBXReader com a lista de contatos persistidos no banco de dados.

Como é possível verificar todos os métodos são tipados com tipos genéricos, assim não temos por obrigação que os parâmetros sejam de um tipo específico como é comum na maioria dos frameworks de persistências.

O TDBXReader é uma classe do DBExpress 4 que fornece um leitor unidirecio-

Listagem 3. Declaração da classe TContato

```
TContato = class(TBase)
public
  property OID: Integer;
  property Nome: String;
  property Email: String;
  property Telefone: String;
end;
```

Listagem 4. Classe de conexão com singleton

```
TDBConnection = class
  strict private
  FConnection: TDBXConnection;
  FCommand: TDBXCommand;
  class var FInstance: TDBConnection;
  constructor CreatePrivate;
  public
  constructor Create;
  class function GetInstance: TDBConnection;
  property Command: TDBXCommand read FCommand;
end;

constructor TDBConnection.Create;
  begin
  raise Exception.Create('Para obter uma instância de TDBConnection
  utilize TDBConnection.GetInatance');
end;

constructor TDBConnection.CreatePrivate;
begin
  FConnection :=
    TDBXConnectionFactory.GetConnectionFactory.GetConnection
    ('DBContato', 'root','root');
  FCommand := FConnection.CreateCommand;
end;

class function TDBConnection.GetInstance: TDBConnection;
begin
  if not Assigned(FInstance) then
  FInstance := TDBConnection.CreatePrivate;
  Result := FInstance;
end;
```

Listagem 5. Script de criação da tabela

```
CREATE TABLE dbcontato.tbl_contato (
  CONTATOID INTEGER NOT NULL AUTO_INCREMENT,
  NOME VARCHAR(50) NOT NULL,
  TELEFONE VARCHAR(20) NOT NULL,
  EMAIL VARCHAR(255) NOT NULL,
  PRIMARY KEY (CONTATOID)
);
```

Nota do DevMan

A versão 4 do framework DBExpress foi totalmente reescrita em Delphi e está 100% orientada a objetos, com dezenas de classes e métodos que facilitaram e muito a vida de quem desenvolve utilizando o conceito de orientação a objetos. Além disso o framework é compatível com a plataforma Win32 e .NET. As principais classes deste framework são a TDBXConnection que tem por função realizar e gerenciar uma conexão com o banco de dados e a classe TDBXCommand que entre outras coisas tem a função de executar comandos SQL de toda natureza no banco de dados.

Nota

Neste artigo assumiu-se o banco de dados MySql para desenvolver o assunto. Utilizando o DBExpress 4 é necessário lançar mão de duas units específicas, a saber: DBXCommon e DbxMySql. A primeira é padrão independente do banco de dados escolhido, já a segunda varia de acordo com o banco que se está trabalhando.

Nota do DevMan

Os padrões de projetos são soluções descritas e catalogadas para problemas no desenvolvimento de softwares orientados a objetos. Eles descrevem soluções testadas e que podem ser reutilizadas sem um números de vezes. Os padrões de projetos mais conhecidos no meio são os 23 definidos pela GOF (gang of four) ou Gangue dos quatro. Entre eles temos o mais simples e eficaz padrão chamado Singleton. Ele é utilizado quando se quer garantir que exista apenas uma instância de um determinado objeto em uma aplicação. Para isso ele utiliza um método de classe que tem por função testar e então decidir se deve instanciar um objeto ou retornar um já criado anteriormente. Este padrão é muito utilizado em classes de conexão com banco de dados, controle de usuários entre outros.

nal para um conjunto de linhas de base de dados. TDBXReader é retornado por TDBXCommand.ExecuteQuery. O método TDBXReader.Next é utilizado para acessar as linhas do result set. Os valores da linha podem ser acessados usando a propriedade Value que é um array. A propriedade Value é sobrecarregada e podemos utilizar tanto a posição da coluna quanto seu nome.

O método GetTableName é utilizado pelos demais métodos, sua implementação é descrita na **Listagem 10** e de uma forma simples é possível notar como o uso do RTTI proporciona extrair informações importantes de um objeto em tempo de execução. Ao criar um objeto do tipo TRttiContext é possível lançar mão do método GetType que efetivamente atribui à variável TypObj todas as informações providas pelo compilador. Feito isso, um loop é realizado nesta estrutura em todos os atributos que ela possui e caso seja encontrado um atributo que é do tipo TableName, através do método exit o método é abandonado retornando o conteúdo do atributo TableName que nada mais é do que o nome da tabela onde o objeto em questão será persistido.

O método Add (**Listagem 11**) tem a função de receber um objeto qualquer e utilizar RTTI para gerar uma instrução de insert no banco de dados. Novamente como no método anterior, um objeto do tipo TRttiContext é criado e em seguida todas as informações do objeto são extraídas para a variável TypObj. Três variáveis do tipo string são utilizadas para a geração da instrução SQL. Para cada propriedade é verificada a existência ou não de atributos e havendo um ou mais atributos em uma propriedade é verificado se este atributo é do tipo KeyField. Caso seja um KeyField a propriedade não é mapeada, pois este campo é auto incremental no banco de dados.

Caso não seja chave primária, o nome da propriedade passa a fazer parte da instrução insert a ser gerada para que em seguida seja feita uma verificação para mapear o tipo da propriedade, que deve ser feita utilizando RTTI. Além disso, todos os valores obtidos devem ser convertidos para string para serem concatenados na montagem da instrução. Após esta eta-

Listagem 6. Classe do atributo TableName, utilizado na persistência

```
TableName = class(TCustomAttribute)
private
 FName: String;
public
 constructor Create(aName: String);
 property Name: String read FName write FName;
end;

constructor TableName.Create(aName: String);
begin
 FName := aName
end;
```

Listagem 7. Classe do atributo KeyField, utilizado na persistência

```
KeyField = class(TCustomAttribute)
private
 FName: String;
public
 constructor Create(aName: String);
 property Name: String read FName write FName;
end;

constructor KeyField.Create(aName: String);
begin
 FName := aName;
end;
```

Listagem 8. Atributos aplicados

```
[TableName('TBL_CONTATO')]
TContato = class(TBase)

[KeyField('CONTATOID')]
property OID: Integer read FOID write SetOID;
```

Listagem 9. Declaração da classe GenericDAO

```
TGenericDAO = class
private
 class function GetTableName<T: class>(Obj: T): String;
public
 class procedure Add<T: class>(Obj: T);
 class procedure Delete<T: class>(Obj: T);
 class procedure Update<T: class>(Obj: T);
 class function GetAll<T: class>(Obj: T): TDBXReader;
end;
```

Listagem 10. Implementação do método GetTableName

```
class function TGenericDAO.GetTableName<T>(Obj: T): String;
var
 Contexto: TRttiContext;
 TypObj: TRttiType;
 Atributo: TCustomAttribute;
 strTable: String;
begin
 Contexto := TRttiContext.Create;
 TypObj := Contexto.GetType(TObject(Obj).ClassInfo);
 for Atributo in TypObj.GetAttributes do
 begin
  if Atributo is TableName then
   Exit(TableName(Atributo).Name);
 end;
end;
```

pa, são removidas as últimas vírgulas adicionadas no processo de montagem já mencionado anteriormente, por último a instrução é gerada e executada no banco de dados utilizando a classe Singleton já criada.

O método Update, exibido na **Listagem 12,** segue os mesmos procedimentos que já foram descritos anteriormente. As diferenças aqui são duas: a primeira é que o comando SQL gerado neste momento é de Update e a segunda é que agora ao identificar que uma das propriedades possui o atributo KeyField ela não será mais desprezada ou "passada", pelo contrário. Ao identificar o atributo KeyField em uma propriedade o conteúdo do atributo é adicionado à variável que será usada na montagem da cláusula where assim como o conteúdo da mesma. As demais propriedades são mapeadas e para cada propriedade encontrada novamente é verificado seu tipo e concatenados nome e valor da propriedade às variáveis utilizadas para gerar o comando SQL de Update.

O método Delete é bem mais simples**,** novamente o objeto passado como parâmetro é mapeado via RTTI. Ao localizar a propriedade que possui o atributo KeyField a mesma é utilizada na cláusula where para que no fim a instrução delete seja executada no banco de dados. Como o método é bastante semelhante aos anteriores, omitimos a implementação aqui, mas você pode encontrar o código completo para download.

Nota do DevMan

O método GetValue que é invocado da classe TRttiProperty retorna um objeto do tipo TValue e, apesar do TValue possuir métodos AsString, AsExtended, AsInteger entre outros, estes métodos não realizam Casting dos tipos, ou seja, é necessário testar o tipo da propriedade para que possa utilizar o método correto para obter seu valor.

Listagem 11. Implementação do método Add

```
class procedure TGenericDAO.Add<T>(Obj: T);
var
    Contexto: TRttiContext;
    TypObj: TRttiType;
    Prop: TRttiProperty;
    strInsert, strFields, strValues: String;
    Atributo: TCustomAttribute;
    IsKey: Boolean;
begin
    strInsert := '';
    strFields := '';
    strValues := '';
    strInsert := 'INSERT INTO ' + GetTableName(Obj);
    Contexto := TRttiContext.Create;
    TypObj := Contexto.GetType(TObject(Obj).ClassInfo);
    for Prop in TypObj.GetProperties do
    begin
     IsKey := False;
     for Atributo in Prop.GetAttributes do
      if Atributo is KeyField then
       IsKey := True;
      if not IsKey then
      begin
       strFields := strFields + Prop.Name + ',';
        case Prop.GetValue(TObject(Obj)).Kind of
        tkWChar, tkLString, tkWString, tkString,
        tkChar, tkUString:
        strValues := strValues +
        QuotedStr(Prop.GetValue(TObject(Obj)).AsString) + ',';
        tkInteger, tkInt64:
        strValues := strValues +
        IntToStr(Prop.GetValue(TObject(Obj)).AsInteger) + ',';
        tkFloat:
        strValues := strValues +
        FloatToStr(Prop.GetValue(TObject(Obj)).AsExtended) + ',';
       else
        raise Exception.Create('Type not Supported');
       end;
      end;
    end;
    strFields := Copy(strFields, 1, Length(strFields) - 1);
    strValues := Copy(strValues, 1, Length(strValues) - 1);
    strInsert := strInsert + '(' + strFields + ') VALUES (' + strValues + ')';
    TDBConnection.GetInstance.Command.Text := strInsert;
    TDBConnection.GetInstance.Command.ExecuteUpdate;
end;
```

Listagem 12. Implementação do método Update

```
class procedure TGenericDAO.Update<T>(Obj: T);
var
  Contexto: TRttiContext;
  TypObj: TRttiType;
  Prop: TRttiProperty;
  strUpdate, strSet, strKeyField, strKeyValue: String;
  Atributo: TCustomAttribute;
  isKey: Boolean;
begin
  strUpdate  := '';
  strSet     := '';
  strKeyField := '';
  strKeyValue := '';
  strUpdate := 'UPDATE ' + GetTableName(Obj);
  Contexto := TRttiContext.Create;
  TypObj := Contexto.GetType(TObject(Obj).ClassInfo);
  for Prop in TypObj.GetProperties do
  begin
   isKey := False;
   for Atributo in Prop.GetAttributes do
    if Atributo is KeyField then
    begin
     strKeyField := KeyField(Atributo).Name;
     strKeyValue := IntToStr(Prop.GetValue(TObject(Obj)).AsInteger);
     isKey := True;
    end;
    if isKey then Continue;
    strSet := strSet + Prop.Name + ' = ';
    case Prop.GetValue(TObject(Obj)).Kind of
     tkWChar, tkLString, tkWString, tkString,
     tkChar, tkUString:
      strSet := strSet +
      QuotedStr(Prop.GetValue(TObject(Obj)).AsString) + ',';
     tkInteger, tkInt64:
      strSet := strSet +
      IntToStr(Prop.GetValue(TObject(Obj)).AsInteger) + ',';
     tkFloat:
      strSet := strSet +
      FloatToStr(Prop.GetValue(TObject(Obj)).AsExtended) + ',';
     else
      raise Exception.Create('Type not Supported');
    end;
  end;
  strSet := Copy(strSet, 1, Length(strSet) - 1);
  strUpdate := strUpdate + ' SET ' +
   strSet + ' WHERE ' + strKeyField + ' = ' + strKeyValue;
  TDBConnection.GetInstance.Command.Text := strUpdate;
  TDBConnection.GetInstance.Command.ExecuteUpdate;
end;
```

Para finalizar a classe GenericDAO falta apenas a implementação do método GetAll (**Listagem 13**). Novamente é lançada mão da classe de conexão com o banco de dados e através dela é executado o comando de seleção. O método ExecuteReader retorna um objeto do tipo TDBXReader, retornado pela função.

Publicando os métodos no servidor

Até o momento todas as estruturas apresentadas são totalmente desacopladas e independentes, isso mostra o quanto uma aplicação orientada a objetos facilita o reaproveitamento de código. A classe TContato não possui vínculo com a classe TGenericDAO e esta por sua vez está apta a mapear qualquer objeto que a ela for passado. Além disso, a classe de conexão que presta serviço para a classe TGenericDAO pode, sem nenhuma restrição, ser utilizada para outro fim neste ou em outro projeto qualquer.

E o mais importante, em nenhum momento até agora foi citada qualquer referência à aplicação multicamadas, ou seja, tudo visto até o momento pode ser utilizado em uma aplicação client-server sem qualquer problema. Mas para que toda a infraestrutura criada possa ser consumida em uma aplicação multicamadas é necessário criar uma interface de comunicação com os clientes. Esta interface na nova arquitetura DataSnap chama-se Server Methods, e é ilustrada na **Listagem 14**. Sua estrutura possui os métodos necessários para permitir que as aplicações clientes possam adicionar um novo contato, remover um contato existente, atualizar um contato ou retornar a lista de contatos cadastrados.

Note que todos os métodos, exceto GetAllContacts, operam sobre um objeto, neste caso do tipo JSONValue. Isso porque a comunicação entre a aplicação cliente e servidora se faz através de notação JSON como citado anteriormente. Daí a necessidade dos métodos de conversão criados na classe TBase. Ao receber um objeto em notação JSON a aplicação servidora terá que transformá-lo em um objeto convencional e persisti-lo utilizando a classe TGenericDAO. Isso é mostrado na **Listagem 15.** A implementação dos métodos é extremamente simples e limpa e toda a inteligência de persistência é encapsulada na classe TGenericDAO.

Listagem 13. Implementação do método GetAll

```
class function TGenericDAO.GetAll<T>(Obj: T): TDBXReader;
begin
 TDBConnection.GetInstance.Command.Text := 'Select * from ' +
   GetTableName(Obj);
 Result := TDBConnection.GetInstance.Command.ExecuteQuery;
end;
```

Listagem 14. Declaração da classe TSMContato

```
{$METHODINFO ON}
 TSMContato = class(TComponent)
 public
   procedure DeleteContact(aObject: TJSONValue);
   procedure UpdateContact(aObject: TJSONValue);
   function GetAllContacts: TDBXReader;
   procedure AddContact(aObject: TJSONValue);
 end;
 {$METHODINFO OFF}
```

Listagem 15. Implementação dos métodos da classe TSMContato

```
procedure TSMContato.AddContact(aObject: TJSONValue);
var
  Contato: TContato;
begin
  Contato := TContato.JSONToObject<TContato>(aObject);
  TGenericDAO.Add<TContato>(Contato);
end;

procedure TSMContato.DeleteContact(aObject: TJSONValue);
var
  Contato: TContato;
begin
  Contato := TContato.JSONToObject<TContato>(aObject);
  TGenericDAO.Delete<TContato>(Contato);
end;

function TSMContato.GetAllContacts: TDBXReader;
begin
  Result := TGenericDAO.GetAll(TContato.Create);
end;

procedure TSMContato.UpdateContact(aObject: TJSONValue);
var
  Contato: TContato;
begin
  Contato := TContato.JSONToObject<TContato>(aObject);
  TGenericDAO.Update<TContato>(Contato);
end;
```

Aplicação Cliente

A aplicação cliente é bem simplória, a **Figura 3** mostra a interface com alguns componentes. O componente TListView é ótimo para ser utilizado em aplicações que interagem com objetos pois, além de permitir que os dados de um objeto sejam listados ao usuário de forma simples e intuitiva, também dá a prerrogativa de associar o ponteiro do objeto para cada item listado. Isso significa que além de exibir os dados do objeto é possível manter seu ponteiro para posterior utilização.

Para conectar ao servidor de aplicação a aplicação cliente necessita de um componente TSqlConnection devidamente configurado utilizando o driver DataSnap do DBExpress. Com a conexão devidamente configurada é possível gerar a classe proxy que faz a interface de comunicação da aplicação cliente com o servidor de aplicação. Essa classe é gerada automaticamente ao selecionar a opção Generate DataSnap Client Classes.

A implementação dos botões New, Update, Delete e Cancel é feita na **Listagem 16** onde basicamente, para os botões que se aplica, é mudado o conteúdo da variável FState declarada na seção private do formulário. É invocado o método ControlButtons, que controla os componentes visuais e o método clear que tem por função limpar o conteúdo dos edits. Ambos são exibidos na **Listagem 17**.

Toda e qualquer conexão, ou melhor, interação com os métodos do servidor de aplicação, deve ser feita através da classe Proxy (**Listagens 18** e **19**). No botão Save é feito o teste da variável FState e dependendo do seu conteúdo é decido qual método será invocado no servidor. Porém, como o que é trafegado entre o cliente e o servidor é sempre um objeto, é criado um objeto do tipo TContato, e, para ele são passados os dados contidos nos edits. Por último, um objeto do tipo TSMContatoClient, que nada mais é do que o proxy de comunicação com o servidor de aplicação, é criado e associado à variável Client.

O método de classe ObjectToJSON é usado para converter o objeto TContato em um objeto JSON, para que então possa ser enviado ao servidor. É importante saber que nos casos de atualização ou exclusão de um objeto é de suma importância que se envie também o ID do objeto que se deseja deletar.

A **Figura 4** mostra a aplicação em execução.

Conclusão

O desafio está lançado, o conceito e a boa prática foram mostrados neste artigo. Uma experiência interessante seria criar novas classes de negócio e submetê-las à classe TGenericDAO para ver como a persistência irá se comportar. Obvio que há outros assuntos a serem abordados e tratados pela classe DAO, como o caso de associações entre classes do tipo composição e agregação, além do fato da otimização dos comandos SQL.

Nota do DevMan

Um proxy, em sua forma mais geral, é uma classe que funciona como uma interface para outra classe. Em padrões de projeto a função de uma classe proxy é prover um representante para outro objeto de modo a controlar o acesso a este. No caso do DataSnap pode-se dizer que há um proxy remoto, que é quando a classe proxy provê um representante local para um objeto em um espaço de endereçamento diferente, neste caso o servidor de aplicação.

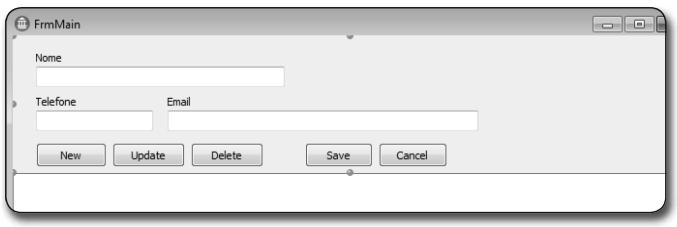


Figura 3. Interface da aplicação cliente

Listagem 16. Implementação dos botões New, Update, Delete e Cancel

```
procedure TFrmMain.btnCancelClick(Sender: TObject);
begin
  FState := tsNone;
  Clear;
  ControlButtons;
end;

procedure TFrmMain.btnNewClick(Sender: TObject);
begin
  FState := tsNew;
  ControlButtons;
  Clear;
end;

procedure TFrmMain.BtnUpdateClick(Sender: TObject);
begin
  FState := tsUpdate;
  ControlButtons;
end;

procedure TFrmMain.BtnDeleteClick(Sender: TObject);
begin
  FState := tsDelete;
  ControlButtons;
end;
```

Listagem 17. Implementação dos métodos ControlButtons e Clear

```
procedure TFrmMain.Clear;
begin
  EdtNome.Clear;
  EdtEmail.Clear;
  EdtTelefone.Clear;
  If EdtNome.CanFocus then
    EdtNome.SetFocus;
end;

procedure TFrmMain.ControlButtons;
begin
  btnNew.Enabled := FState = tsNone;
  BtnUpdate.Enabled := (FState = tsNone) and
    (ListView1.Selected <> nil);
  btnSave.Enabled := (FState <> tsNone);
  btnDelete.Enabled := (FState = tsNone) and
    (ListView1.Selected <> nil);
  EdtNome.Enabled := (FState <> tsNone);
  EdtTelefone.Enabled := (FState <> tsNone);
  EdtEmail.Enabled := (FState <> tsNone);
end;
```

Listagem 18. Implementação do botão Save

```
procedure TFrmMain.btnSaveClick(Sender: TObject);
var
  Client: TSMContatoClient;
  Contato: TContato;
  M: TJSONMarshal;
  JSON: TJSONValue;
begin
  try
    Contato := TContato.Create;
    Contato.Nome := EdtNome.Text;
    Contato.Telefone := EdtTelefone.Text;
    Contato.Email := EdtEmail.Text;
    Client := TSMContatoClient.Create(DBServer.DBXConnection);
    case FState of
      tsNew:
      begin
        JSON := TContato.ObjectToJSON<TContato>(Contato);
        Client.AddContact(JSON);
      end;
      tsUpdate:
      begin
        Contato.OID := StrToInt(ListView1.Selected.Caption);
        JSON := TContato.ObjectToJSON<TContato>(Contato);
        Client.UpdateContact(JSON);
      end;
      tsDelete:
      begin
        Contato.OID := StrToInt(ListView1.Selected.Caption);
        JSON := TContato.ObjectToJSON<TContato>(Contato);
        Client.DeleteContact(JSON);
      end;
    end;
  finally
    FreeAndNil(Client);
    FreeAndNil(Contato);
    ShowReader;
    Clear;
    ControlButtons;
  end;
end;
```

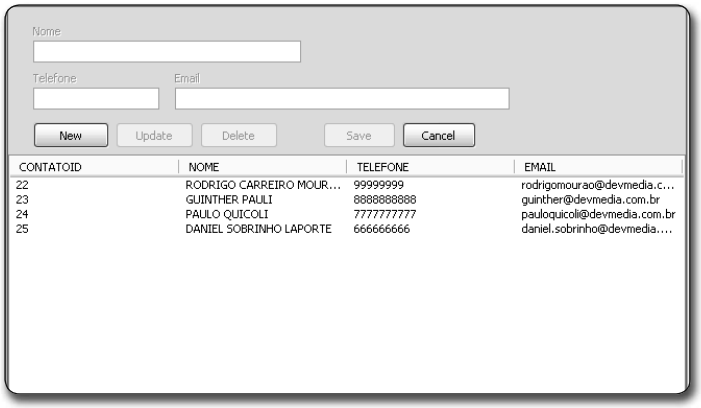


Figura 4. Aplicação em execução

O que foi mostrado aqui é apenas um pontapé inicial, mostrando o que é e como é possível aplicar alguns padrões de desenvolvimento multicamadas na nova arquitetura DataSnap. Outras tecnologias podem ser agregadas como, por exemplo, criar um servidor totalmente RESTFull permitindo assim que não somente aplicações Win32 possam consumir nossos métodos no servidor de aplicação mas também qualquer aplicação Web que tenha suporte a Java Script por exemplo.

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br
http://rodrigocarreiro.com.br
http://twitter.com/carreiromourao
http://radphp.com.br

Sócio diretor da RM Factory Soluções em TI, empresa especializada em consultoria e desenvolvimento utilizando soluções RAD da Embarcadero. Instrutor certificado pela Borland e Delphi Product Certified. Editor técnico da Revista ClubeDelphi, administrador da Comunidade RAD PHP, membro ativo na comunidade e apaixonado pelo Delphi.

Listagem 19. Implementação do ShowReader

```
procedure TFrmMain.ShowReader;
var
 I: Integer;
 LColumn: TListColumn;
 LRowItem: TListItem;
 aReader: TDBXReader;
 Client: TSMContatoClient;
begin
 ListView1.Clear;
 ListView1.Columns.Clear;
 ListView1.Update;
 Client := TSMContatoClient.Create(DBServer.DBXConnection);
 aReader := Client.GetAllContacts;
 for I := 0 to aReader.ColumnCount - 1 do
 begin
  LColumn := ListView1.Columns.Add;
  LColumn.Caption := aReader.ValueType[I].Name;
  LColumn.AutoSize := True;
 end;
 while aReader.Next do
 begin
  LRowItem := ListView1.Items.Add;
  for I := 0 to aReader.ColumnCount - 1 do
  begin
   if I = 0 then
    LRowItem.Caption := aReader.Value[I].AsString
   else      LRowItem.SubItems.Add(aReader.Value[I].AsString);
  end;
 end;
 ...
end;
```

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No novo Fórum da DevMedia você vai encontrar canais específicos de Delphi, ASP.NET, Java, Banco de Dados e Engenharia de Software; além de ter contato com profissionais qualificados da área para troca de informações, sugestões e muito mais.

ACESSE AGORA
www.devmedia.com.br/forum

DevMedia

FinalBuilder
Automação de Builds no

Desenvolver softwares vai muito além de boas práticas, conhecer bem seu IDE, aproveitar seus recursos, saber analisar, corrigir, melhorar, prevenir entre tantas outras coisas. Sem contar os fatores extras que exigem que se tenha conhecimento sobre outras coisas não relacionadas necessariamente a desenvolvimento de software, como informações sobre Sistema Operacional, segurança e bancos de dados por exemplo. Desenvolver projetos é quase uma arte e certamente pode exigir muito trabalho e muito tempo. Para aumentar qualidade, produtividade, trabalho colaborativo e agilidade os desenvolvedores podem sempre optar por ferramentas ou recursos extras que podem fornecer uma "mãozinha" no desenvolvimento.

A nova versão do Delphi, chamada de XE, trouxe consigo muitas das melhores ferramentas do mercado "embutidas" no seu IDE. A ideia nesta nova versão é justamente esta, tornar seu ambiente ainda mais produtivo utilizando o conceito RAD com ferramentas que tornem seu trabalho mais ágil e com o máximo de qualidade possível.

Entre estas novidades da nova versão pode-se facilmente destacar o FinalBuilder (tema deste artigo), AQTime para realização de Profiling de código, Subversion para controle e versionamento de código-fonte e o CodeSite para Debugs altamente avançados. Além das novidades não relacionadas a ferramentas que tratam do suporte a mais nova onda tecnológica do momento: a computação nas nuvens, representada pela plataforma em nuvem da Microsoft, o Windows Azure.

O FinalBuilder como já dito anteriormente, é uma ferramenta que realiza o processo de build automatizado. Falando assim pode parecer um simples software que utiliza o compilador do Delphi em background para fazer um build de um projeto seu, porém, ele vai muito além desta funcionalidade.

Em um ambiente de desenvolvimento comum, este processo de gerar o build final do projeto (liberação da versão oficial) não costuma ser diário, mas sim, semanal, quinzenal ou até mesmo mensal, dependendo de alguns fatores, como a ocorrência de algum bug que tenha passado despercebido pelos casos de teste (o que muda o fluxo e faz com que se tenha a necessidade de liberação de uma versão mais rapidamente). Outro fator é a grande quantidade de modificações nos projetos, que principalmente para uma grande equipe é claro, é algo comum. E é aí que entra o FinalBuilder. Com ele é possível realizar uma compilação automática de todo o seu código-fonte e de qualquer projeto (não só desenvolvido em Delphi). Seria possível desenvolver um compilador automático utilizando o próprio compilador dcc32.exe (do Delphi) e passando instruções via linha de comando, através do prompt de comandos por exemplo, ou até mesmo desenvolvendo uma interface para isto, porém o limite seria apenas o compilador do Delphi, e é claro que um trabalho desses demanda muito tempo pra fazer apenas um pouco do que o FinalBuilder se propõe a fazer, e melhor, automaticamente.

Resumo DevMan

De que trata o artigo:

Lançado recentemente pela Embarcadero, o Delphi XE trouxe consigo várias ferramentas de terceiros do mercado. Neste artigo será abordada uma delas, o FinalBuilder.

Para que serve:

O FinalBuilder serve para facilitar o processo de distribuição dos builds de projeto, provendo toda a preparação do ambiente, agendando o build, realizando validações no seu projeto entre tantas outras funcionalidades.

Em que situação o tema é útil:

Em qualquer tipo de aplicação onde se deseja adicionar organização ao modo de compilação dos projetos. Grandes ou pequenas equipes podem tirar proveito destes recursos, presentes não só para o compilador do Delphi como para tantos outros.

Final Builder:

O processo de build de um software pode variar de empresa para empresa. Contudo o build não se resume a apenas compilar um programa. Várias outras tarefas podem estar envolvidas no processo, tarefas essas que consomem tempo. O FinalBuilder permite automatizar essas tarefas e assim agilizar todo o processo.

Por que utilizar uma ferramenta de build automático?

Antes de conhecer o FinalBuilder é interessante justificar o porquê de se utilizar uma ferramenta como esta e quais benefícios ela pode oferecer a uma equipe de desenvolvedores e grandes projetos, ou até mesmo para casos de pequenas equipes com projetos menores.

Delphi XE

DANIEL SOBRINHO LAPORTE

Em projetos de grande porte é comum encontrar facilmente mais de 1000 units, incluindo formulários, classes e bibliotecas, DLLs incorporadas, arquivos de registro, arquivos de configuração, arquivos de recursos, drivers, arquivos de fontes, aplicações cliente, aplicações servidoras, testes, instalador e muitas outras coisas. O FinalBuilder simplesmente organiza e controla todo o processo de maneira automatizada e customizada, permitindo que se tenha o total domínio sobre o compilador Delphi a ponto de informar o que deve ser feito antes de realizar o build, ou caso haja uma falha de compilação, quais são os passos que devem ser levados em conta antes e após a geração dos arquivos necessários. Pois então, o FinalBuilder possibilita exatamente essa experiência de flexibilidade em um projeto, permitindo priorizar determinadas etapas, rodar testes para saber se tudo deu certo, atualizar as informações do executável e ainda interagir com o ambiente externo a partir de outras aplicações de terceiros, e o melhor, sem ter que necessariamente programar.

Nota do DevMan

É possível realizar compilações por linha de comando utilizando o compilador padrão do Delphi, o dcc32.exe, localizado no diretório BIN da instalação padrão. Muitos aplicativos já utilizam este recurso durante o próprio processo de instalação, como o caso da suíte de componentes JediVCL, que realiza durante seu processo de Install o build e execução do seu próprio projeto (.dprj). O arquivo install.bat ilustra claramente o processo realizado pelo compilador:

```
:: compile installer

echo [Compiling installer...]

cd install\JVCLInstall

if EXIST JVCLInstall.cfg  del JVCLInstall.cfg

..\..\packages\bin\dcc32ex.exe --requires-jcl -Q -B
-DJVCLINSTALLER -DUSE_DXGETTEXT -E..\..\bin -I.;..\..\
common -U..\..\common;..\..\run -n..\..\dcu JVCLInstall.dpr
```

Nota-se que o compilador dcc32 foi embutido dentro da pasta bin do diretório packages. A compilação anterior passa alguns parâmetros para o projeto JVCLInstall.dpr. É possível realizar uma série de parametrizações via linha de comando, tornando o seu processo de build transparente neste caso, ou mesmo fazendo com que o seu executável seja "auto-compilável" a partir disso. Para saber sobre a lista de parâmetros disponíveis basta digitar dcc32 no prompt de comando e a lista será exibida.

Nota do DevMan

Profiling é uma técnica que consiste em inspecionar/investigar o código-fonte e detectar possíveis pontos falhos, problemas entre outras coisas, sendo muito utilizada principalmente na área de engenharia de software. Na versão XE esta tarefa foi designada à ferramenta AQTime que é totalmente integrada ao IDE do Delphi através do menu de mesmo nome.

Automatizar seu processo de build pode proporcionar um ganho de produtividade considerável, já que uma vez configuradas as etapas do projeto, elas permanecem gravadas e prontas para uso. Sem contar na possibilidade de se automatizar ferramentas de terceiros como o Inno Setup para gerar o instalador do projeto.

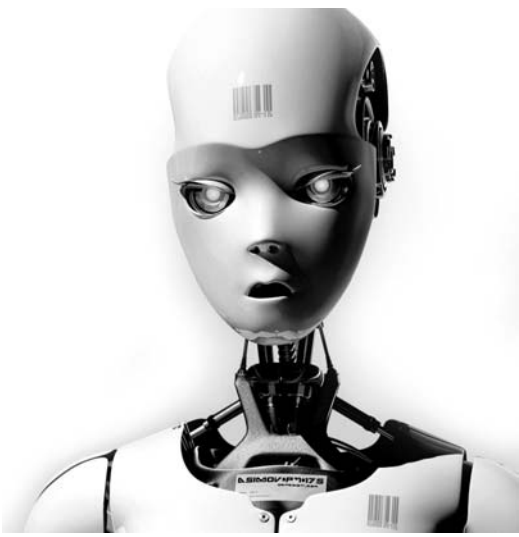
Com o FinalBuilder simplesmente se cria um projeto e se adicionam ações que devem ser tomadas para a geração do executável do projeto. É possível personalizar este build de inúmeras formas, criando validações através de comandos condicionais, utilizando ações "pré-definidas" ou criando suas próprias ações, integrando com outros aplicativos como SVN por exemplo, agendar seu build, visualizar logs de erro de build, tratamentos de exceção e muito mais.

Conhecendo o FinalBuilder

Pode-se dizer que realizar compilações comuns com o FinalBuilder é relativamente um processo bastante simples, porém, à medida que se necessita de mais recursos do IDE, a complexidade aumenta um pouco, o que exige um certo tempo para entender sua forma de funcionamento por completo. A melhor forma de obter o domínio da ferramenta é utilizando os próprios arquivos de exemplo disponíveis na instalação do FinalBuilder, outras duas fontes de informação são os links de demonstrativos e help do FinalBuilder, disponíveis na sessão Links ao final do artigo.

Abordando inicialmente o ambiente do FinalBuilder, este basicamente é composto por três partes (**Figura 1**). A primeira parte exibe a navegação e informações na qual abrangem um overview do projeto de build como um todo. No centro encontra-se a janela principal, responsável por ilustrar a lista de ações de acordo com a ação programada, acompanhamento do build e de cada uma das ações, progresso de compilação etc. No rodapé a janela de log, que como sugere, exibe detalhadamente tudo que ocorreu com o projeto, inclusive possíveis avisos de Warning para casos de ações que podem resultar em possíveis erros ou problemas de compilação.

Na mesma aba onde existe um "Object Inspector" do FinalBuilder, fica a aba de ações. Esta aba contém um dos principais



requisitos da ferramenta, que são justamente as ações a serem configuradas no projeto. Para que se entenda melhor o conceito das ações, o ideal é imaginar que estas são instruções pré-definidas que serão responsáveis pelas ações do projeto. Estas ações podem ser funções, uma estrutura condicional de comandos como if/else ou a interação com uma outra aplicação qualquer.

Primeiramente é mostrado um exemplo simples, sem complexidade alguma e no decorrer são adicionadas novas funcionalidades que abrangem outros recursos. O ambiente do FinalBuilder é intuitivo e simples de trabalhar.

Action Lists

Pode-se dizer facilmente que as Actions são um dos principais conceitos responsáveis por "dar vida" ao FinalBuilder, porém é interessante destacar a possibilidade de realizar tarefas alternativas ao processo de compilação comum, mesmo sem saber como utilizar as variáveis.

As Action Lists são caracterizadas por ser um grupo de ações. Com elas é possível não só ter uma organização maior do projeto, mas sim, tomar controle da ocorrência das tarefas do processo de compilação. Quando se inicia um projeto no FinalBuilder, por padrão, são criados alguns tipos de Action Lists, como Main e OnFailure (**Figura 2**). A Action List Main corresponde à ação principal de execução do processo de compilação, ou seja, quando se inicia um projeto simples, devem ser configuradas as ações principais que serão executadas no processo de compilação. A Action List OnFailure funciona como uma espécie de evento da Action List Main onde caso ocorra algum tipo de erro, poderão ser configuradas ações específicas de erro para que o FinalBuilder faça após a falha, como envio de e-mails, execução de algum programa externo, entre outras coisas.

Embora o FinalBuilder permita a criação de ações próprias, sempre convém dar uma olhada na lista de ações já disponíveis. Existem mais de 200 previamente criadas que necessitam apenas de uma simples configuração para agregar funcionalidades à compilação.

Para a criação de um simples projeto basta configurar as informações a partir do modo de busca de ação (**Figura 3**). É importante que a aba de ações do lado esquerdo do IDE esteja selecionada para poder adicionar as ações referentes ao compilador desejado.

Antes de adicionar as ações é extremamente importante atentar-se na Action List selecionada pois, caso se engane de ActionList sua execução não será bem sucedida. Após selecionar o Build do Delphi Win32 na ActionList Main, a janela de configuração de projeto é exibida (**Figura 4**). As configurações dessa janela são bem semelhantes às opções do IDE do Delphi. Atentando as opções pode-se perceber que nada mais é do que um mix entre as opções de ambiente (Environment Options) e as opções de projeto do menu de mesmo nome. É interessante atentar ao fato de que é possível selecionar as opções configuradas no projeto, ou sobrescrevê-las, permitindo a configuração de componentes, BPLs entre outros recursos diretamente pelo FinalBuilder. Outro fator no mínimo interessante é que é possível realizar a compilação de um projeto desenvolvido em qualquer versão do IDE (desde que esteja instalado).

Configurando um projeto comum como o da **Figura 4**, é possível realizar o processo de compilação apenas clicando no botão Run (▶) ou pelo atalho F9. Aqui vale destacar que os projetos criados pelo FinalBuilder também podem ser debugados assim como no Delphi ou no Visual Studio. Há também como realizar o debug das ações configuradas, marcando a ação

Figura 1. FinalBuilder

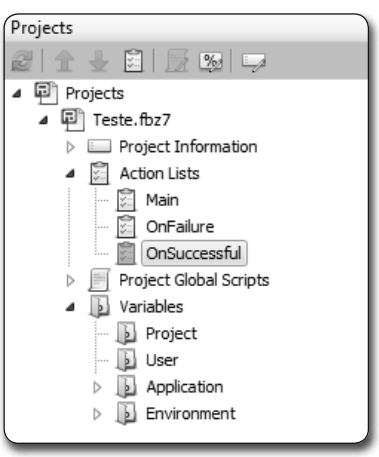


Figura 2. Criando uma nova Action List personalizada no FinalBuilder e visualizando as ações padrões

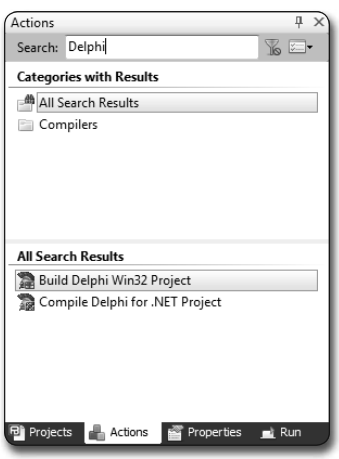


Figura 3. Utilizando a busca de ações do FinalBuilder

com um Breakpoint ou simplesmente pressionando F5 (também como no Delphi). Os itens selecionados para Debug ficam marcados com um quadrado vermelho.

Ao compilar projetos, o lado esquerdo do FinalBuilder exibe o progresso da compilação e o status de sucesso ou falha do projeto. No centro é exibida uma lista das ações que estão sendo executadas, bem como o status de cada uma delas, indicando se a execução daquela determinada ação definida ocorreu conforme o esperado ou não, possibilitando a identificação exata da ação que não foi bem sucedida. A parte inferior do IDE exibe as informações de Log do projeto como tempo de execução da ação, e a sequência de compilação das ações (**Figura 5**).

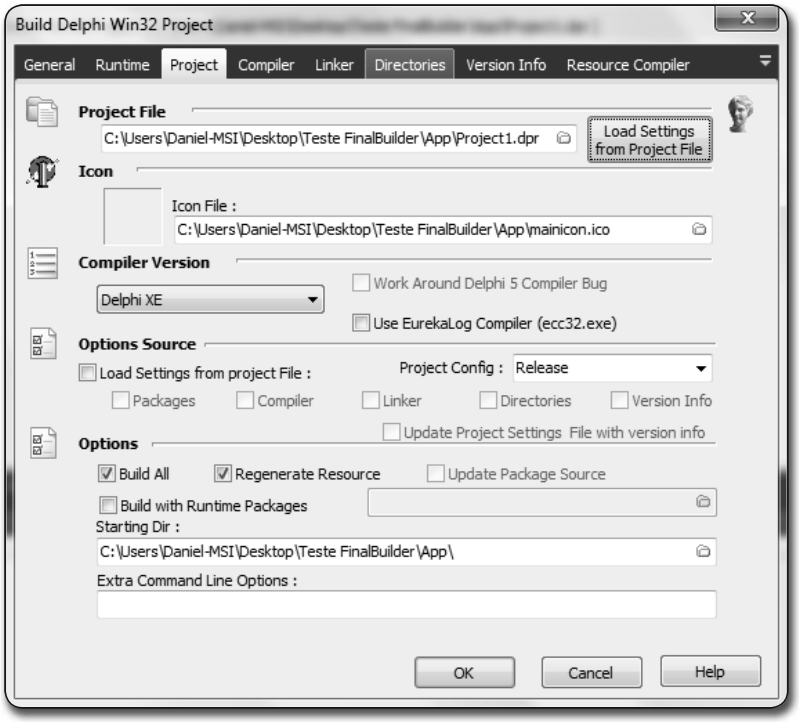


Figura 4. Carregando um projeto Delphi Win32 existente usando o compilador do Delphi XE

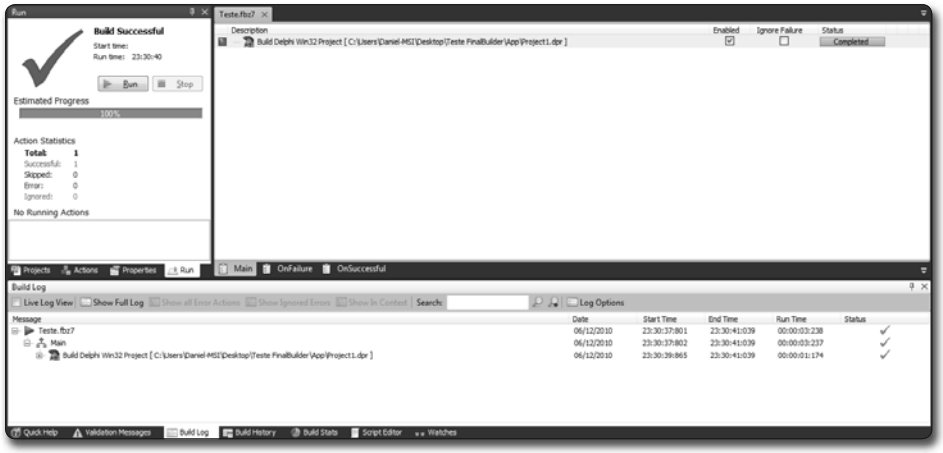


Figura 5. FinalBuilder em funcionamento

Controle de fluxo

Aprofundando um pouco mais nos conceitos do FinalBuilder, temos o Flow Control ou controle de fluxo. A grande sacada deste conceito é a capacidade de desviar o fluxo de compilação dos projetos, permitindo não só uma interatividade como um controle total do que é feito. Com isto pode-se facilmente reproduzir o que é feito no desenvolvimento de qualquer sistema, ou seja, reproduzir os conceitos de estruturas de controle através de laços definidos (for) ou indefinidos (while), comandos condicionais (if/else), comandos de opções (cases) e o mais interessante de todos, o tratamento de erros.

O controle de tratamento de erros é feito através dos comandos de captura das linguagens tradicionais. É possível formular sintaxes comuns ao Delphi como Try Finally e Try Catch ("Exception").

A partir destes conceitos é possível realizar tarefas condicionais. Um exemplo mais simples possível do conceito de captura é a realização de logs, onde a aplicação trata o processo de compilação do projeto principal no bloco try e em caso de erros a estrutura catch captura o erro, gravando um arquivo .txt no diretório da aplicação e exibindo o erro que pode ter sido armazenado dentro de uma variável do próprio FinalBuilder.

Outra aplicação do fluxo de controle pode ser realizada através de uma condicional a partir da leitura de uma variável. Por exemplo, uma mesma aplicação que possua fontes diferentes (um servidor de aplicação em COM+, um em BSS e um em DataSnap) neste caso, pode ser lido um valor armazenado em registro, arquivo texto ou .ini onde é definido qual modelo será compilado e, durante o processo de compilação, este valor pode ser lido e alimentado dentro de uma variável no FinalBuilder. Os testes condicionais desta variável (if/else) decidem qual será o projeto que deve ser compilado, em caso de erro pode-se gerar o Log em arquivo e enviar um e-mail a equipe, notificando o processo de erro. O grande detalhe é que estas regras configuradas uma vez, o processo de compilação pode ser agendado e todo o trabalho poupado, sendo realizado automaticamente pelas configurações de agendamento.

Nota

A palavra reservada Catch corresponde ao Exception nas linguagens Object Pascal e D# (Delphi Prism), porém, no FinalBuilder a palavra reservada Exception não existe, deste modo deve ser adotada a palavra Catch para a captura e tratamento de erros.

Nota do DevMan

Diretivas de compilação são instruções particulares do compilador que fazem com que o mesmo seja capaz de desviar seu modo de compilação padrão, realizando ações específicas ou mesmo selecionando a instrução mais apropriada com a configuração. Este é um conceito muito utilizado dentro do IDE para manter a compatibilidade entre as diferentes versões que são lançadas e a compatibilidade entre a CLX. É comum ver este tipo de instrução em componentes, aonde são adotadas justamente para manter a retro compatibilidade.

As **Figuras 6** e **7** ilustram exatamente o processo descrito anteriormente, onde há a compilação do projeto principal e a criação de um arquivo texto de log no diretório da aplicação no caso de falhas. Na **Figura 6** é possível perceber que o fluxo de compilação é bem sucedido e neste caso o bloco de exceção (catch) é ignorado (skipped). Já na **Figura 7** o erro ocorre, a ação em que ocorreu o erro tem seu status alterado e o bloco de exceção é acionado.

Variáveis

Assim como no conceito de variável de um projeto qualquer, as variáveis no FinalBuilder têm o propósito de armazenar valores. Dentro do FinalBuilder existem quatro tipos distintos de variáveis, são eles:

• Projeto (Project): São variáveis específicas para o projeto criado. Uma vez criadas só estarão disponíveis naquele projeto específico do FinalBuilder;

• Usuário (User): Assim como as variáveis de projeto estas são particulares de projeto e por usuário, ou seja, pode-se tomar determinadas ações de acordo com o usuário logado na máquina;

• Aplicação (Application): Estas variáveis são definidas pelo FinalBuilder logo na inicialização. As variáveis disponíveis aqui dependem de quais softwares estão instalados na máquina. Vale lembrar que os valores que estão definidos aqui nestas variáveis não podem ser modificados, sendo disponíveis apenas para leitura (Read-Only);

• Ambiente (Environment): Estas já são velhas conhecidas dos desenvolvedores. As variáveis de ambiente estão presentes em Sistemas Operacionais, instalações de bancos de dados e até no próprio Delphi. Um exemplo é a variável WINDIR que especifica o diretório em que o Windows se encontra instalado na máquina. É importante destacar que deve-se tomar certo cuidado com este tipo de variável, pois caso você crie uma variável deste tipo e modifique o seu ambiente, você provavelmente vai ter que recriá-la.

Vale lembrar que as variáveis de projeto e usuário podem ser disponibilizadas como variáveis de ambiente no momento de sua criação. Outro detalhe importante é que o FinalBuilder restringe a criação de variáveis com nomes idênticos mesmo sendo de tipos diferentes.

Para qualquer operação a ser realizada com variáveis deve-se obrigatoriamente utilizar o caractere "%" no começo e no final, por exemplo: %WINDIR%. Por padrão, as variáveis do FinalBuilder são definidas como Variant, devido a comunicação deste tipo com tipos primitivos ser mais fácil de ser manuseada, porém nada impede de definir tipos primitivos.

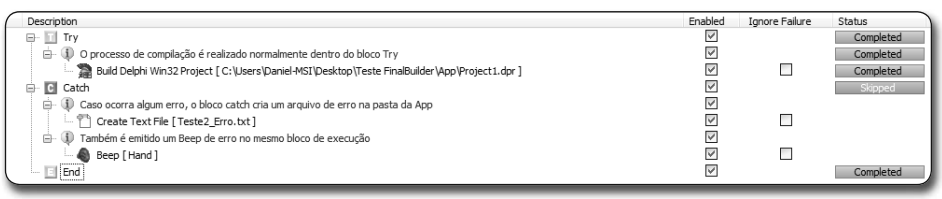


Figura 6. Compilação bem sucedida do bloco Try

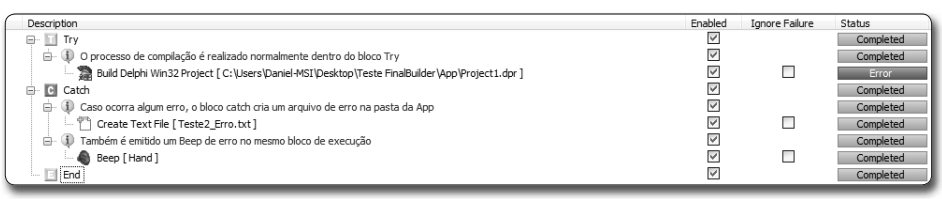


Figura 7. Compilação mal sucedida resultando na execução do bloco Catch

Assim, utilizando os conceitos de fluxo de controle, ações e variáveis é possível realizar praticamente qualquer tarefa com variáveis. Um exemplo interessante para sua aplicação seria verificar se o diretório de geração do Build já existe, se não existir o FinalBuilder o cria e realiza a compilação, caso contrário apenas realiza o processo de compilação. A **Figura 8** ilustra um exemplo de aplicação de variáveis onde é checada a existência de um diretório emitindo um beep de confirmação caso exista ou um beep de erro caso contrário.

Enviando e-mails

Enviar e-mails pode ser uma das tarefas mais interessantes no processo de automatização da compilação. No cotidiano de uma grande equipe é comum o controle de versionamento do código-fonte realizando o commit das alterações pendentes, rodar testes, organizar e separar DLLs, compilar aplicações cliente e servidoras, separar banco de dados (em alguns casos disponibilizar o servidor de banco de dados juntamente com a aplicação), criar o instalador, atualizar a versão do instalador e do Build, preparar arquivos de configuração entre tantas outras tarefas. Após realizadas tarefas como essas em muitos casos a versão do instalador é disponibilizada em um FTP para só então permitir os clientes realizarem o download da atualização. Em casos como estes se pode ter projetos separados para realizar o build das aplicações principais e o build dos testes. Em equipes distintas esta comunicação do progresso entre uma etapa e outra geralmente é realizada por e-mail. Neste caso o próprio FinalBuilder dá uma mãozinha se encarregando de enviar um e-mail para a equipe de suporte, para a equipe de testes e até mesmo para o cliente ao disponibilizar o executável final e oficial. É perfeitamente possível enviar e-mails em caso de falha, para que o tempo de correção seja minimizado, fazendo com que o processo se torne mais ágil.

Basicamente o processo de configuração do envio de e-mails é semelhante à configuração de um gerenciador de e-mail comum como Outlook ou Mozilla Thunderbird por

exemplo. A grande jogada nesta técnica é configurar a ação Send Mail na Action List correta e para os casos específicos de sucesso ou falha. Vale lembrar que é possível configurar o envio para qualquer tipo de conta. Caso a conta de e-mail não esteja configurada corretamente, os Warnings do que há de errado aparecem na exibição do log. A **Figura 9** ilustra o processo de envio e recebimento do e-mail configurado.

Compilações agendadas

O tempo para a geração de um build completo pode demorar certo tempo considerável, já que várias tarefas podem estar envolvidas. Outro fator é que estas compilações não são realizadas diariamente, geralmente há um cronograma pré-definido com a compilação ao final da semana ou ao final do mês (ou as famosas "compilações noturnas"), havendo modificações nesse cronograma apenas em caso de erros, como surgimento de bugs que passaram despercebidos nos testes. Para isso o FinalBuilder pode realizar todos estes procedimentos descritos de forma automática e agendada, poupando assim um membro da equipe de desenvolvimento, testes e é claro, conseguindo muito mais agilidade em todo o processo.

A opção Scheduler possibilita o agendamento não só de um, mas de vários projetos simultaneamente. O mais interessante disto tudo é que o FinalBuilder não necessita estar aberto para que sejam realizadas as compilações agendadas, basta que existam agendamentos e todo o resto é feito por conta própria.

É disponibilizado um Wizard com as opções para configuração do projeto. Neste Wizard selecionam-se o projeto, a periodicidade da compilação (diária, semanal, mensal entre outras) juntamente com o horário e data de início do agendamento realizado. Um detalhe importante aqui é que se deve informar o usuário e senha de administrador do computador em que será feita a compilação agendada. Isto ocorre devido a questões de permissões. O ideal é que se tenha ao menos um usuário com privilégios de administrador e senha previamente configuradas, pois caso contrário, este processo não funcionará corretamente. Por último, é sugerido que se selecione o modo de compilação que pode ser feito pelo próprio IDE ou através de linhas de comando. Vale lembrar que isto ocorrerá em background.

Além destas opções ainda existem algumas configurações avançadas que permitem a customização de acordo com modo de gerenciamento de energia (para notebooks por exemplo) ou até mesmo por período de ociosidade. A **Figura 10** ilustra uma Task já devidamente configurada e pronta para ser executada. Após este processo o FinalBuilder pode estar fechado e a compilação na qual você configurou será executada, realizando as ações principais do processo de compilação e em caso de exceções notificando a equipe.

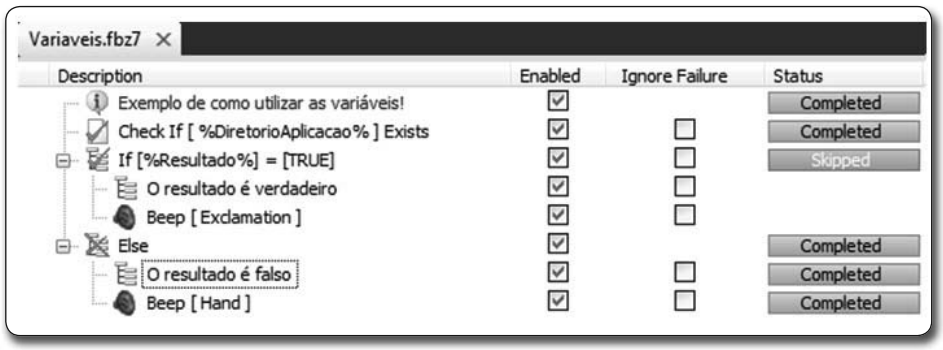


Figura 8. Realizando compilações com variáveis

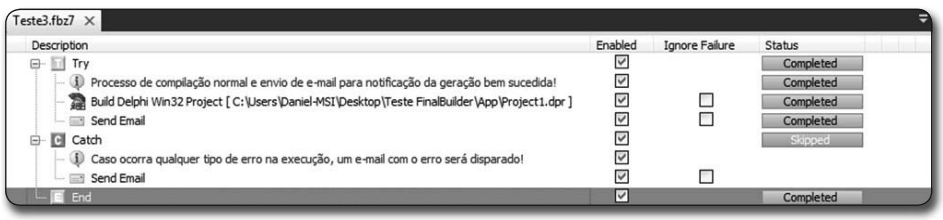


Figura 9. Enviando e-mails no processo de compilação

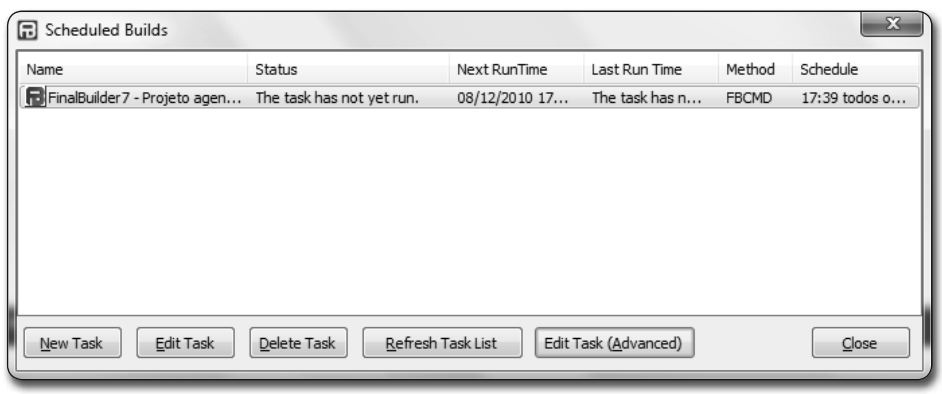


Figura 10. Criando uma Task de compilação automática com o FinalBuilder

Nota do DevMan

Ao trabalhar com um software de controle de versionamento de código (SVN), é comum ouvir os termos Check-in, Check-Out, Update e Commit, opções que também podem ser usadas no FinalBuilder. Basicamente estas são as opções disponíveis para se trabalhar com um repositório de fontes:

Check-in: É responsável por criar o repositório oficial dos fontes do sistema no servidor;

Check-out: Cria um repositório e uma cópia do código-fonte do servidor na máquina local;

Update: Verifica a versão de código-fonte local e a do servidor em busca de diferenças ou modificações. Caso haja modificações em ambos os locais de uma mesma unit, o SVN identifica que há um conflito, exibindo as linhas que sofreram alterações e sugerindo sincronizar (merge), ou considerar uma das duas (a local ou a do servidor);

Commit: Envia as alterações do código-fonte local para o código-fonte oficial (servidor).

Nome da Ação	Pasta	Descrição
Read Ini File	Registry & Ini	Lê um valor de um arquivo .Ini. O valor lido pode ser reutilizado usando variáveis.
Write Ini File	Registry & Ini	Grava um valor em arquivo Ini. O valor pode ser algo obtido pelo próprio ambiente do FinalBuilder.
Erase Ini File	Registry & Ini	Apaga um arquivo Ini.
Read Registry Value	Registry & Ini	Lê um valor de uma chave de registro do Sistema Operacional.
Set Registry Value	Registry & Ini	Grava um valor de uma chave de registro do Sistema Operacional.
Delete Registry Key/Value	Registry & Ini	Deleta um valor de uma chave de registro do Sistema Operacional.
ST Check In	StarTeam	Realiza o processo de Check In dentro utilizando a interatividade com o StarTeam.
ST Check Out	StarTeam	Realiza o processo de Check Out utilizando a interatividade com o StarTeam.
Burn CD/DVD	CD/DVD Burning	Permite gravação de CD/DVDs.
Create ISO Image	CD/DVD Burning	Permite a criação de arquivos de imagem tipo ISO.
COM + Register	COM +	Registra uma dll/servidor COM+ (Muito utilizado em ambientes multicamadas anteriores ao novo DataSnap)
COM + Delete	COM +	Deleta uma dll/servidor COM+ (Muito utilizado em ambientes multicamadas anteriores ao novo DataSnap.
ADO Execute SQL	Database	Permite a execução de um script no banco de dados necessitando apenas da configuração da String de conexão.
Send Email	Internet	Envia e-mails pré-programados.
FTP Connect	Internet	Conecta-se a um diretório FTP.
FTP Upload	Internet	Realiza o Upload de algo para o FTP.
FTP Download file	Internet	Realiza o Download de algo do FTP.
FTP Change Directory	Internet	Permite a mudança de diretório dentro do FTP.
Build Inno Setup Script	Install Builders	Monta um instalador do Inno Setup a partir de um arquivo de script (.iss).
Subversion Add	Subversion	Realiza o processo de adição de arquivo ao SVN.
Subversion Checkout	Subversion	Realiza o processo de checkout do repositório SVN.
Subversion Update	Subversion	Realiza o processo de Update de fonte no repositório do SVN.
Execute Program	Windows OS	Executa um programa no processo de Build.
If COM Class Registered	Windows OS	Verifica se há uma classe registrada baseando-se na Guid.
Shell Execute	Windows OS	Assim como no Delphi, o Shell Execute permite a execução de programas apenas pelo caminho/nome.
Create Shortcut	Windows OS	Cria um atalho.

Tabela 1. Lista das principais ações disponíveis no FinalBuilder

"Mais ação"

Uma das grandes vantagens da utilização de ações no FinalBuilder é a possibilidade de integração entre o ambiente do IDE e o ambiente externo. Com isso, é possível realizar praticamente tudo que necessitar no processo de compilação. Outro fator interessante é a compatibilidade entre outros Softwares da Borland como o StarTeam, por exemplo, ou mesmo com softwares que compõem o novo pacote do RAD Studio XE como o SubVersion (SVN). A **Tabela 1** ilustra algumas das ações disponíveis no FinalBuilder, e dá uma ideia da dimensão e poder da ferramenta. Vale lembrar que são mais de 200 ações para integração com o processo de Build.

Aqui podemos ter uma ideia do que é possível fazer. Por exemplo, podemos instruir o FinalBuilder para compilar um projeto inteiro, se tudo sair OK fazer o Check-in das fontes para o StarTeam / Subversion, subir a versão do projeto compilado para um FTP para que possa ser baixado pelos clientes, ou ainda gravar um DVD. Ou ainda gerar uma instalação do build final. E finalmente enviar um email para os responsáveis pela equipe de desenvolvimento.

Conclusão

Como vimos o FinalBuilder é uma excelente opção para automatizar seus processos de build de projeto, permitindo o gerenciamento, organização e controle de seus projetos de uma maneira inteligente. É uma excelente opção para quem possui dificuldade em distribuição do seu projeto, ou mesmo realiza alterações frequentes, pois poderá através desta ferramenta gerenciar todos os processos, ganhando muita produtividade, facilidade na distribuição e geração do executável final que irá para o cliente.

Daniel Sobrinho Laporte
daniel.sobrinho@devmedia.com.br
daniel.laporte@gmail.com
www.twitter.com/daniel_laporte
Colunista e Editor Técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

Demos do FinalBuilder
http://www.finalbuilder.com/demos.aspx

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

Faça a renovação de sua assinatura agora mesmo.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Padrão Strategy

Encapsule responsabilidades criand

Quando se está criando um software, a tendência é torná-lo o mais reutilizável possível, ou seja, minimizar o trabalho necessário para fazer alterações ou implementar novas funções, mas existem muitos problemas que dificultam o trabalho do programador de tornar o software reutilizável. Um exemplo destes problemas é o acoplamento entre as classes do sistema, pois sem o conhecimento adequado o programador provavelmente fará um programa cujas classes estarão fortemente interligadas.

Alto acoplamento e baixa coesão são dois problemas que acontecem muito com sistemas mal projetados. Acoplamento é o nome que se dá para a ligação existente entre duas classes, mas não no sentido de relacionamento, mas sim, no que uma conhece da outra. Quando uma classe conhece demais sobre outra, esta torna-se dependente da outra e vice-versa. Assim, se essa outra sofre alteração, é muito provável que a primeira também tenha que se ajustar em relação à mudança sofrida. Portanto, um nível grande de acoplamento não é recomendado.

A coesão por sua vez pode ser traduzida como a objetividade que uma classe pode possuir. É muito comum, mas não ideal, encontrar classes que realizam várias operações, por exemplo: imprimir um relatório e exibir uma mensagem na tela. Uma baixa coesão pode causar problemas durante a manutenção do sistema, visto que uma classe que faz várias coisas ao ter seu código alterado pode implicar em falhas e outras funcionalidades.

Um padrão de projeto é uma solução para problemas como o citado anteriormente. Claro que se pode inventar uma solução própria, mas até que ponto vale a pena investir tempo para criar uma solução para um problema que já foi solucionado?

Resumo DevMan

De que trata o artigo:

Refatoração de código aplicando princípios de orientação a objetos que facilitam a implementação das soluções chamadas de padrões de projetos, implementado o padrão Strategy para definir grupos de classes com comportamentos diferentes que podem ser trocados em tempo de execução.

Para que serve:

Tornar projetos de software mais flexíveis, baseando-se nos princípios da orientação a objetos, para também que se tornem reutilizáveis. Os padrões de projetos são uma maneira de usar os princípios da orientação para resolver problemas complexos do desenvolvimento de software.

Em que situação o tema é útil:

No desenvolvimento de sistemas orientados a objetos, em projetos onde os requisitos possuem uma constante mudança e qualidade for uma meta a s er alcançada.

Padrão Strategy:

Algumas pessoas sentem dificuldade para usar padrões de projetos e construir designs elegantes nos seus sistemas, pois não conseguem encontrar situações para usá-los, apesar de ter o conhecimento teórico sobre alguns padrões. Isto é comum porque os problemas existentes em um software dificilmente são solucionados apenas com um padrão de projeto. Por isto você precisa encher sua cabeça com vários deles para então ser capaz de criar uma solução completa usando os padrões para criar um bom design. Fazendo uma analogia a um programa de computador, os padrões de projetos funcionam como drivers que devem ser carregados para o seu cérebro para que você possa usá-los quando necessário. Esta é maneira mais fácil de conseguir aplicá-los, por tanto quanto mais padrões diferentes você conhecer mais facilidade terá para solucionar um problema com eles.

Além do mais quando um padrão de projeto é implementado, um projeto não está sendo prejudicado, pois um padrão de projeto não é simplesmente uma solução, ele normalmente é a melhor solução, uma vez que estão embasados em princípios de programação orientada a objetos.

Os princípios de orientação a objetos são como regras de um protocolo que devem ser seguidos para que se alcance um bom design de projeto – os princípios ensinam o que é bom e o que deve ser evitado em um projeto de software. Conhecer pelo menos alguns destes princípios já é um começo se o objetivo é dominar a técnica de programação orientada a objetos.

Um bom design de software

O bom design de um software possui três características reconhecidas:

Manutenção – É a facilidade com que um sistema pode ser alterado para aplicar novas regras de negócio exigidas, mas não só regras e sim também melhorar seu desempenho, corrigir falhas e outras solicitações mais. Aplicações que possuem um bom design exigem poucos recursos (tempo, esforço e até mesmo capital) para realizar sua devida manutenção.

Reusabilidade – Em um sistema adequado é possível que os seus componentes possam ser reutilizados por outros sem grandes dificuldades.

o grupos de classes especializadas

Rafael Stavarengo

Robustez- É a estabilidade do sistema em situações extremas do tipo: tratamento de entradas erradas por parte de usuários, carga máxima de dados em determinado momento do dia.

Enquanto isso, um design ruim também apresenta os seguintes problemas:

Rigidez – Dizemos que um sistema é rígido quando uma alteração nele é difícil. E essa alteração é considerada difícil porque rotinas internas estão, muito provavelmente, entrelaçadas a um ponto onde alterar qualquer coisa leva a uma cascata de alterações. Nessas situações quando o sistema fica grande, é difícil estimar o prazo para uma alteração já que todo o impacto deve ser medido antes.

Fragilidade – Por ter rotinas muito ligadas umas às outras, a alteração em uma leva a problemas nas outras. Quem aqui nunca alterou algo em um sistema que depois causou um problema em uma parte que parecia não estar relacionada com a alteração feita? Corrigir esse tipo de problema pode levar a outros, desencadeando uma leva de ajustes que custam tempo e que consequentemente custa dinheiro. Um sistema nessa situação perde sua credibilidade, pois vivem contenção de erros, impedindo que seus desenvolvedores mantenham uma qualidade futura ao projeto.

Reusabilidade – Neste caso, a falta de. Quando for preciso reutilizar alguma rotina, o trabalho pode ser tão difícil que é menos custoso desenvolver novamente essa rotina.

Refatoração

O projeto apresentado no artigo é uma aplicação para o usuário enviar emails rapidamente. Para usá-lo, é necessário informar o nome de usuário e a senha para conexão com o servidor de email. Depois o usuário escreve a mensagem propriamente dita e por fim informa os destinatários e envia o email. Toda a lógica necessária para esta função está dentro do único método da aplicação. Aparentemente um sistema simples, porém seu código apresenta problemas estruturais que serão identificados e corrigidos através de refatoração.

Refatoração é a ação de modificar um projeto a fim de torná-lo mais flexível diante das mudanças do dia a dia. As alterações feitas durante este processo normalmente não são visíveis ao usuário, ou seja, a refatoração não adiciona novos comportamentos no projeto, ela apenas modifica o código (se necessário) para torná-lo mais flexível e reutilizável.

Em "Refatoração: Aperfeiçoando o Projeto de Código Existente", Martin Fowler - mentor da tecnologia de objetos - desmitifica práticas importantes e demonstra como os desenvolvedores de software podem tirar vantagens reais da técnica de refatoração. O autor demonstra que é possível tornar um projeto ruim num projeto de código robusto e bem projetado. Fowler fornece um passo a passo explicando como aplicar as mais de 70 técnicas de refatoração existentes no livro. Os exemplos ilustrativos estão escritos em Java, mas as ideias são aplicáveis a qualquer linguagem orientada a objetos.

Interfaces

Mais uma vez vamos falar neste artigo sobre este que sem dúvida é um dos pilares da POO. Interfaces - São um dos mais avançados recursos da POO e permitem mais um nível de abstração na análise. O Delphi suporta interfaces desde a versão 3, que na época foram desenvolvidas principalmente para integração com o COM. Interfaces ajudam a separar o desenvolvimento das classes ou módulos entre vários programadores, inclusive de empresas diferentes. Basta que as interfaces sejam combinadas ou convencionadas corretamente, como um contrato, e que tenham métodos com escopo, objetivos, entradas e saídas bem definidos.

Isso permite padronizar as interações e trocas de mensagens entre objetos, e permite também criar objetos de linhagens diferentes que obedeçam as mesmas interfaces, para que possam ser intercambiáveis. Num contexto dinâmico onde as regras de negócio podem ser modificadas

Nota do DevMan

Princípios SOLID - Essa sigla é um acrônimo para a palavra em inglês solid, que significa sólido, solidez. Cada letra representa um acrônimo de um princípio, ou seja, temos um acrônimo de um acrônimo onde: S de Single Responsability Principle (SRP), O de Open/Closed Principle (OCP), L de Liskov Substitution Principle (LSP), I de Interface Segregation Principle (ISP) e D de Dependency Inversion Principle (DIP). Esse conjunto é o resultado de estudos e experiências de vários desenvolvedores que foram coletados e organizados formando boas práticas básicas para um bom sistema orientado a objetos. Os princípios de modelagem de objetos são guias que levam a construir software flexível e estável. Evitando dependência excessiva entre as classes, duplicação de código e outros problemas mais.

ou o produto ser totalmente remodelado antes mesmo de ser posto em produção, o uso de interfaces garante uma grande flexibilidade. Separando conceitos em partes encaixáveis é possível mapear facilmente quais implementações modificar, para atender uma regra de negócio sem comprometer um contrato, e quais interfaces devem ser estendidas através de herança. Além dos objetivos mencionados, as interfaces também servem para simular herança múltipla.

Nota

É dito que um código está "cheirando mal" quando está mal escrito, com problemas visíveis e que precisa de uma intervenção, ou seja, precisa ser reescrito.

Listagem 1. O código de uma aplicação mal projetada

```
01  procedure TFormulario.BotaoEnviarClick(Sender: TObject);
02  var
03    Smtp: TIdSMTP;
04    Email: TIdMessage;
05  begin
06    Progresso.Lines.Clear;
07    Progresso.Lines.Add('Preparando para enviar');
08    Email := TIdMessage.Create(nil);
09    Email.From.Name := NomeDoRemetente.Text;
10    Email.From.Address := EmailParaRespsota.Text;
11    Email.Recipients.EMailAddresses := EmailDestinatario.Text;
12    Email.Subject := Assunto.Text;
13    Email.Body.Text := Mensagem.Text;
14    Smtp := TIdSMTP.Create(nil);
15    Smtp.Host := SmtpHost.Text;
16    Smtp.Port := Porta.Value;
17    if UsarSsl.Checked then
18    Smtp.AuthType := satSASL
19    else
20    Smtp.AuthType := satDefault;
21    Smtp.OnStatus := SmtpStatus;
22    try
23      try
24        Smtp.Authenticate;
25        Smtp.Connect;
26        Smtp.Send(Email);
27      except
28        on E: Exception do
29          Progresso.Lines.Add('ERRO: ' + E.Message);
30      end;
31    finally
32      if Smtp.Connected then
33        Smtp.Disconnect;
34      FreeAndNil(Smtp);
35      FreeAndNil(Email);
36      Progresso.Lines.Add('Fim da rotina de envio.');
37    end;
38  end;
39  procedure TFormulario.SmtpStatus(ASender: TObject;
    const AStatus: TIdStatus;
40    const AStatusText: string);
41  begin
42    Progresso.Lines.Add(AStatusText);
43  end;
```

Identificando falhas

O projeto SendMail possui apenas um método e nele está centralizada toda a funcionalidade da aplicação (**Listagem 1**). O procedimento de enviar e-mail não é complicado, mas o faz parecer que é. Isto porque está muito longo, cheio de responsabilidades e com diversos caminhos alternativos (ifs, case, try), ou seja, o código está "cheirando mal".

Papo de responsabilidade

Como identificar as partes de uma aplicação que precisa ser refatorada? A resposta está nos princípios da programação orientada a objetos. É preciso procurar por partes do código que podem ser aprimoradas por um ou mais princípios, por este motivo é apresentado agora o princípio de responsabilidade única. Se existisse um princípio pré-requisito para conhecer os demais, este seria o tal. Ele diz que toda classe deve ter apenas um objetivo, em outras palavras, toda classe deve ter uma única razão para mudar, ou ainda, toda classe deve ter apenas uma única responsabilidade. Para começar a refatorar o projeto SendMail, é necessário procurar por lugares que não estão de acordo com o princípio de responsabilidade única.

O projeto SendMail atualmente possui apenas um método chamado BotaoEnviarClick, que fica na classe TFormulario. O fato de toda a lógica do projeto SendMail estar em um único método aumenta as chances de este método estar sobrecarregado de responsabilidades e de fato ele está. Algumas de suas responsabilidades: criar e configurar um objeto de TIdMessage – como pode ser visto entre as linhas 8 e 13; criar e configurar um objeto TIdSMTP – como pode ser visto entre as linhas 14 e 21; estabelecer a conexão, autenticar o usuário e enviar o email – como pode ser visto entre as linhas 22 e 37; manter o formulário TFormulario atualizado mostrando o progresso da conexão com servidor de e-mail – como pode ser visto nas linhas 6, 7, 21, 29, e no método que inicia na linha 39.

Depois de identificar as responsabilidades da classe TFormulario já é possível começar a pensar em separá-las em outras classes para que no final exista uma série delas, mas que cada uma tenha apenas uma responsabilidade, assim como TFormulario deverá ter. Isto significa que cada uma das responsabilidades identificadas devem ser separadas da classe TFormulario e colocadas em outras classes.

As alterações

Depois de pronto, por melhor que esteja o programa, é comum que sempre cheguem alterações: as alterações são a melhor forma de descobrir se o projeto está flexível. Então para provar que o projeto SendMail foi mal projetado e que isto aumenta a dificuldade de fazer manutenção no sistema, vamos tentar implementar uma nova função no projeto.

Um dos usuários do projeto SendMail precisa que os e-mails sejam armazenados em arquivos textos ao invés de serem enviados pela internet. Quantas mudanças serão necessárias fazer no projeto por causa desta alteração? Se precisar mudar muita coisa em um projeto por causa de uma alteração, isto é um indicativo de que o projeto está inflexível e que as partes estão fortemente interligadas. No caso do projeto SendMail, uma solução mais rápida para esta alteração é "copiar/colar" o código que aparece na **Listagem 1**, criando outro método, apenas para mudar as instruções que aparecem entre as linhas 22 e 37.

Duplicar o método BotaoEnviarClick com certeza é a solução mais rápida, porém, com o passar do tempo o código duplicado tornaria o projeto cada vez mais difícil de alterar, criando novos bugs e fazendo o programador perder mais tempo em alterações futuras já que qualquer alteração na funcionalidade de enviar

e-mail implicaria em uma mudança em dois métodos.

A única coisa que realmente muda em relação ao código da **Listagem 1** é que para poder salvar os e-mails em arquivos ao invés de enviá-los pela internet as instruções entre as linhas 22 e 37 são alteradas. É entre estas linhas que acontece a conexão e autenticação com o servidor de email e depois o envio do e-mail propriamente dito, por tanto, para evitar a duplicação de código e tornar o sistema mais flexível, é necessário separar a responsabilidade de enviar e-mails da classe TFormulario.

Essa separação é feita por dois motivos: 1) se não houver separação, código será duplicado; 2) enviar e-mail não é a verdadeira responsabilidade da classe TFormulario, o que levanta duas questões importantes: se enviar e-mails não é responsabilidade da classe TFormulario, de quem seria então?

Separando as responsabilidades

No tópico "Papo de responsabilidade" foram levantadas pelo menos quatro responsabilidades diferentes para a classe TFormulario, e nenhuma delas é a verdadeira responsabilidade desta classe. Ela é uma classe ligada a um formulário, ou seja, TFormulario estende de TForm – o formulário controlado por esta classe pode ser visto na **Figura 1**. Portanto a verdadeira responsabilidade da classe TFormulario é controlar o formulário ligado a ela. Coisas como desabilitar o botão "Enviar mensagem" para evitar mais de um clique seguido, é tarefa desta classe, pois diz respeito ao controle das ações do formulário. Todas as coisas feitas por esta classe que não dizem respeito somente ao formulário que ela controla, são consideradas uma responsabilidade extra.

Uma das responsabilidades extras desta classe é o controle sobre a conexão com servidor de e-mail para enviar a mensagem do usuário. O simples fato de esta responsabilidade estar embutida no local errado (na classe TFormulario) dificulta a implementação do novo comportamento de salvar as mensagens em um arquivo texto. Da maneira como está, para se conseguir isto muito código seria duplicado. Então a melhor solução é separar esta responsabilidade extra da classe TFormulario em outra classe. Ao final desta alteração deverão existir duas classes para controlar o envio de e-mail: uma delas envia o e-mail pela internet e a outra salva o e-mail em um arquivo local.

Para fazer esta separação, uma nova classe chamada TTransporteViaSmtp será criada com a única responsabilidade de enviar e-mails pela internet. Esta nova classe receberá o endereço SMTP e as credenciais do usuário para login – feito isto, basta invocar o método Enviar da classe para que o e-mail seja enviado pela internet. No diagrama de classe da **Figura 2** o método Enviar da classe TTransporteViaSmtp recebe um objeto de TIdMessage – este é o objeto com as informações do e-mail que será enviado.

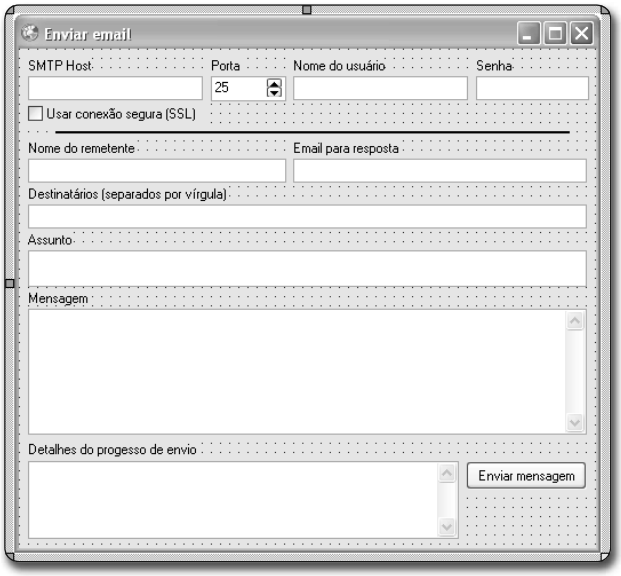


Figura 1. Formulário principal – classe TFormulario

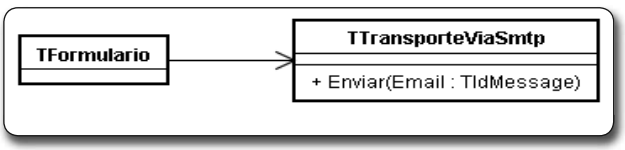


Figura 2. Diagrama de classes depois de separar o envio de e-mail pela internet da classe de formulário

Na **Listagem 2** está parte do código da nova classe criada: TTransporteViaSmtp. Seu construtor espera os parâmetros necessários para estabelecer a conexão com o servidor. Isto significa que para criar um objeto desta classe, é obrigatório informar o endereço do servidor SMTP, a porta de conexão e as credenciais do usuário para login.

No método Enviar da classe TTransporteViaSmtp, o objeto Smtp criado e configurado no construtor é usado para conectar com o servidor e enviar o e-mail. Ao remover essa responsabilidade do método BotaoEnviarClick, seu código ficou mais enxuto e simples de entender. Ficou mais fácil definir qual o objetivo do método. Essa melhoria foi alcançada porque se começou a respeitar o princípio de responsabilidade única que diz que todas as classes de um projeto devem ter no máximo uma responsabilidade.

Agora que o código ficou um pouco menos confuso e mais flexível já é possível implementar o novo comportamento pedido: salvar os e-mails em um arquivo local. Este novo comportamento precisa também de uma nova classe, pois como é sabido, toda classe deve ter apenas uma única responsabilidade.

Nota do DevMan

Componentes Indy – Formam uma biblioteca open source que oferece suporte para o desenvolvimento de clients e servidores TCP, UDP e RAW sobre vários tipos de protocolos. Alguns desses protocolos são: SMTP, POP3, NNTP, HTTP e muito mais. Além da versão disponível para Delphi, atualmente também existe sua versão para o .NET Framework. Com essa biblioteca é possível fazer aplicativos de gerenciamento de e-mails, mensagens instantâneas e até mesmo um servidor web.

Enviar não, salvar localmente

Salvar os e-mails em um arquivo texto é uma nova responsabilidade que o sistema precisa ter, e portanto deverá existir uma nova classe capaz de cumprir esta responsabilidade, essa nova classe é a TTransporteArquivoTexto. A nova classe receberá o caminho do arquivo, e quando o método Enviar for invocado, as informações do e-mail serão salvas no arquivo ao

invés de serem enviadas para o servidor de e-mail. A **Listagem 3** exibe o principal método da classe.

O maior trabalho está no método Enviar. Neste método um arquivo texto é aberto (ou criado caso não exista) e as informações do e-mail são adicionadas neste arquivo. Desta maneira o e-mail é salvo localmente ao invés de ser enviado pela internet. Vale a pena ressaltar que esta nova classe poderia ser melhorada, pois ela também possui mais de uma responsabilidade. O método Enviar além de saber como escrever as informações em um arquivo, também sabe como converter as informações do e-mail para um formato amigável para o usuário ler.

Agora o método BotaoEnviarClick precisa ser atualizado para usar a nova classe criada. Além disto também será necessário adicionar um TCheckBox no formulário para o usuário escolher se deseja enviar o e-mail pela internet ou apenas salvá-lo em um arquivo local. A **Listagem 4** mostra a parte do método BotaoEnviarClick que foi alterada.

As alterações feitas no método desde a sua última versão podem ser vistas. Nesse trecho um if foi adicionado para criar o objeto de transporte que corresponda ao que usuário escolheu.

O novo comportamento que foi pedido já está pronto: agora o projeto SendMail é capaz de enviar e-mails pela internet ou salvá-los localmente. Não parece bom usar explicitamente as classes de transporte dentro da classe TFormulario, pois não é responsabilidade dela saber qual classe deve ser usada para enviar o e-mail. Este trecho precisa ser avaliado melhor.

Listagem 2. A classe com responsabilidade de enviar e-mail pela internet

```
01  procedure TTransporteViaSmtp.Enviar(Email: TIdMessage);
02  begin
03   try
04    Smtp.Authenticate;
05    Smtp.Connect;
06    Smtp.Send(Email);
07   finally
08    if Smtp.Connected then
09     Smtp.Disconnect;
10   end;
11  end;
```

Listagem 3. Uma nova responsabilidade = Uma nova classe (salvando e-mails localmente)

```
01.  procedure TTransporteArquivoTexto.Enviar
     (Email: TIdMessage);
02.   var
03.    Arquivo: TextFile;
04.  begin
05.    Assign(Arquivo, FileName);
06.    if FileExists(FileName) then
07.      Append(Arquivo)
08.    else
09.      Rewrite(Arquivo);
10.    Try
11.    //código de gravação de texto no arquivo
12.    finally
13.      CloseFile(Arquivo);
14.    end;
15.    end;
```

Listagem 4. Usando a nova classe TTransporteArquivoTexto

```
01  try
02    if ArmazenarLocalmente.Checked then
03      TTransporteArquivoTexto.Create('C:\Emails.txt').
        Enviar(Email)
04    else
05      TTransporteViaSmtp.Create(SmtpHost.Text, Porta.
        Value, UsarSsl.Checked).Enviar(Email);
06   finally
07    Progresso.Lines.Add('Fim do algoritmo para enviar e-mail');
08  end;
```

Programe para interfaces

A **Figura 3** mostra o design de uma aplicação que provavelmente trará problemas quando precisar ser alterada: a classe TFormulario depende das classes de transporte de e-mail. Isto aconteceu devido às instruções que podem ser na **Listagem 4**. Nestas linhas a classe TFormulario cria explicitamente objetos das classes de transporte. Se algo mudar nas classes de transporte, a probabilidade da classe TFormulario também precisar mudar é muito alta devido a forte dependência existente entre elas. Isto deve ser evitado, pois uma vez que o transporte dos e-mails não é responsabilidade da classe TFormulario, ela não tem que ser atualizada quando uma das classes de transporte for alterada.

Dizemos que as classes A e B estão fortemente interligadas quando uma depende da outra de tal forma que quando uma delas muda a outra provavelmente também sofrerá alterações. Isto é o oposto do que se espera para um projeto flexível. Existe um princípio da programação orientada a objetos que diz para programarmos sempre para interfaces para que as classes do projeto possam estar interligadas, mas sem que uma seja influenciada pela outra como acontece no projeto SendMail, onde a classe TFormulario é influenciada pelas classes de transportes

Definindo uma interface

O problema do projeto SendMail é que ele não foi programado para interface, mas sim para as implementações. "Interface" aqui não refere-se a interface de interação com o usuário. Neste caso se refere às interfaces dos objetos. Ao observar a **Figura 3** é claro que as classes de transporte têm algo em comum – todas elas têm um método Enviar. Este algo em comum é a interface das classes de transporte. É provável que toda classe de transporte que possa existir no futuro terá o mesmo desenho da classe

de transporte da **Figura 2**, ou seja, terão o método Enviar. É como se estas classes fizessem parte de um grupo especial que serve apenas para enviar e-mails quando o método Enviar é invocado. Porém atualmente não existe nada que determine que estas classes façam parte de um grupo especial de classes – elas apenas existem como qualquer outra classe.

Usar classes abstratas ou interfaces é uma maneira de definir uma categoria de classes. Sendo assim a interface ITransporte (**Listagem 5**) é criada e deve ser implementada por todas as classes que desejam ser consideradas como uma classe de transporte. Uma interface é como se fosse uma classe comum, exceto que os métodos da interface não são implementados, apenas declarados. Assim toda classe que quiser implementar uma interface, deve obrigatoriamente implementar os métodos declarados na interface. A interface ITransporte declara um método chamado Enviar, o que significa que todas as classes que quiserem ser considerada uma classe de transporte deverão obrigatoriamente implementar este método.

A atualização do método BotaoEnviarClick pode ser vista na **Listagem 6**.

Uma nova variável chamada Transporte foi declarada e é do tipo ITransporte – a interface criada na **Listagem 5**. Agora a instância da classe de transporte é criada usando a variável Transporte e é usada para invocar o método Enviar. O benefício do uso de interfaces pode ser visto, pois não se sabe para qual objeto a variável Transporte está apontando – pode ser tanto TTransporteArquivoTexto como também TTransporteViaSmtp – e mesmo assim o método Enviar pode ser invocado porque estas classes implementam a interface ITransporte. Embora as classes TTransporteArquivoTexto e TTransporteViaSmtp já possuam o método Enviar, elas ainda não são consideradas classes de transportes pelo compilador porque não implementaram a interface ITransporte. Para isso sua declaração deve ser ajustada como mostra o código a seguir:

```
TTransporteArquivoTexto = class(TInterfacedObject,
ITransporte)
```

Listagem 5. A interface ITransporte

```
01  unit Transporte;
02  Interface
03  uses IdMessage;
04  Type
05   ITransporte = interface
06     procedure Enviar(Email: TIdMessage);
07    end;
08  Implementation
09  end.
```

Listagem 6. Tentando diminuir a dependência das classes de transporte

```
01 try
02  if ArmazenarLocalmente.Checked then
03   Transporte := TTransporteArquivoTexto.Create
      ('C:\Emails.txt')
04  else
05   Transporte := TTransporteViaSmtp.Create(SmtpHost.Text,
      Porta.Value, UsarSsl.Checked);
06  Transporte.Enviar(Email);
07 Finally
08  Progresso.Lines.Add('Fim do algoritmo para enviar e-mail');
09 end;
```

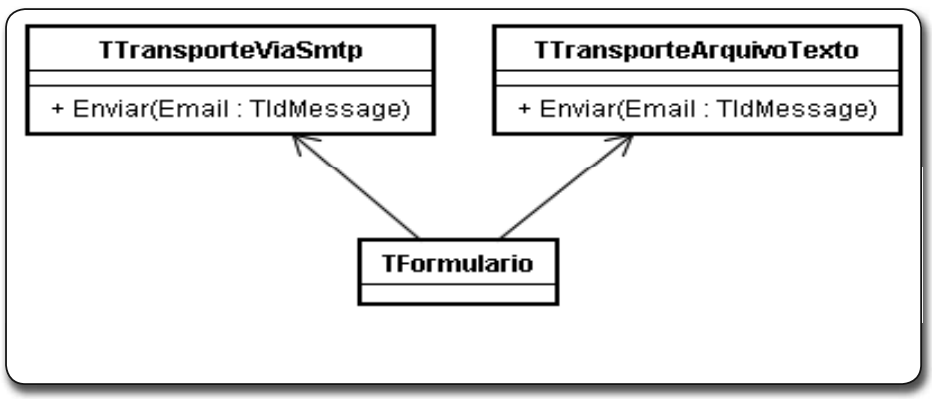


Figura 3. Classes fortemente interligadas

Sempre que quiser implementar uma interface, sua classe deverá estender de TInterfacedObject. TInterfacedObject é um tipo especial de classes desenvolvida para ser a classe base de outras classes que desejam implementar alguma interface. Essa classe já possui em seu código rotinas que tratam do uso correto de interfaces. Objetos que implementam interfaces e são descendentes de TInterfacedObject não necessitam ser destruídos, pois têm um contador de referências interno que automaticamente destrói o objeto que a implementa quando esse contador chega a zero. Além disso esta classe TInterfacedObject tem um mecanismo para trabalhar com multithread que previne que um objeto seja destruído quando ainda não terminou de ser criado. Então, teoricamente usando interfaces e criando classes como descendentes de TInterfacedObject não existe preocupação com a destruição dos objetos e nem com Memory Leak. Um Memory Leak, ou vazamento de memória, é o termo dado para um objeto que foi criado em tempo de execução, contudo, após seu devido uso e descarte, não é destruído – eliminado da memória.

Ao programar para interfaces o projeto diminui as dependências entre as classes, pois ao invés de uma classe depender de várias outras, ela passa a depender apenas da interface que as classes implementam.

Nota

Princípio de programação orientada a objetos – "Programe para uma interface, não para uma implementação".

A **Figura 4** mostra que ao programar para interfaces a classe TFormulario passou a depender somente da interface ITransporte ao invés de depender de todas as classes de transporte (as implementações). Mas algo ainda está errado. Mesmo que o diagrama da **Figura 4** mostre que TFormulario depende somente da interface, no código parece que não é bem assim. A **Listagem 6** mostra que a classe TFormulario continua usando (e dependendo) das implementações da interface ITransporte, que são: TTransporteArquivoTexto e TTransporteViaSmtp. Sendo assim, ainda falta alguma coisa para conseguir tornar a classe TFormulario totalmente independente das classes de transporte.

Interface ou classe abstrata

Quando o princípio diz "programe para interfaces" ele não está se referindo às entidades do tipo interface, pois também é possível seguir este princípio usando classes abstratas. O que este princípio tenta passar é que se deve evitar programar fazendo com que as classes fiquem dependentes umas das outras. A única forma de evitar isto é criar alguma entidade (classe abstrata ou interface), que identifique um grupo de classes com alguma característica comum. Para o projeto SendMail, ter usado uma classe abstrata teria o mesmo resultado, sendo assim como saber quando usar interfaces ou classes abstratas quando precisar definir um grupo de classes?

A diferença mais evidente entre uma classe abstrata e uma interface é que a interface não permite implementação de nenhum método declarado nela. Com uma classe abstrata pode-se tanto criar métodos com implementação quanto métodos sem implementação que deverão obrigatoriamente ser implementados pelas subclasses. Com base nisto, se for necessário criar um grupo de classe cujos métodos serão todos diferentes entre as subclasses, é melhor usar interface. Por outro lado se existir um grupo de classes onde um ou mais métodos se repetem entre as classes do grupo, pode-se usar a classe abstrata para fazer a implementação.

Para um entendimento melhor a **Figura 5** mostra um diagrama do padrão Observer.

Este padrão determina dois grupos de classes: o grupo de classes que observa e o grupo de classes que são observadas. O grupo de classes que observa foi definido usando uma interface que se chama IObservador. Foi escolhida usar uma interface para este grupo, pois o método Notificação é sempre diferente para as classes deste grupo, ou seja, cada implementação de IObservador fará uma coisa diferente quando o método Notificação for invocado.

Por outro lado foi escolhida usar um classe abstrata para definir o grupo de classes que são observadas, pois os métodos AddObservador, RemoveObservador e NotificarObservador nunca mudam entre as subclasses. O único método que muda entre as subclasses é o método chamado MetodoAbstrato. Na verdade este último método não faz parte do padrão Observer – foi colocado no diagrama apenas para justificar a necessidade da classe TObservavel ser abstrata, ou seja a **Figura 5** não mostra um diagrama fiel ao diagrama do padrão Observer.

No caso do projeto SendMail foi escolhido usar uma interface para definir o grupo de classes de transporte, pois o método Enviar é sempre diferente entre as classes do grupo, como acontece no exemplo do grupo de classes observadoras da **Figura 5**.

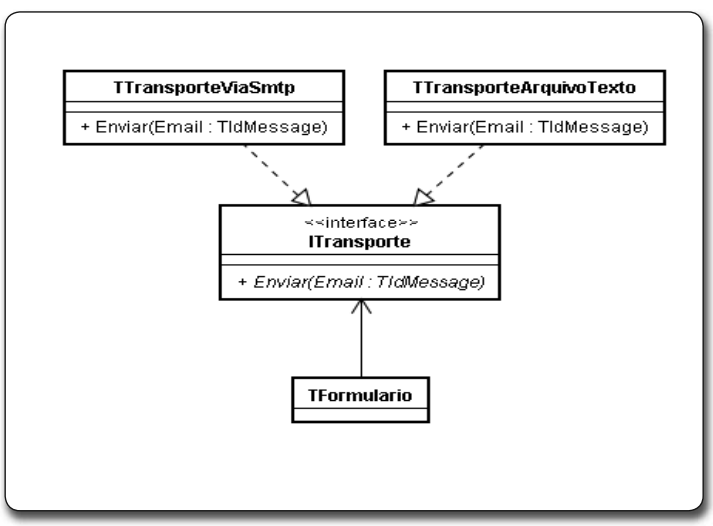


Figura 4. Classes com baixa dependência

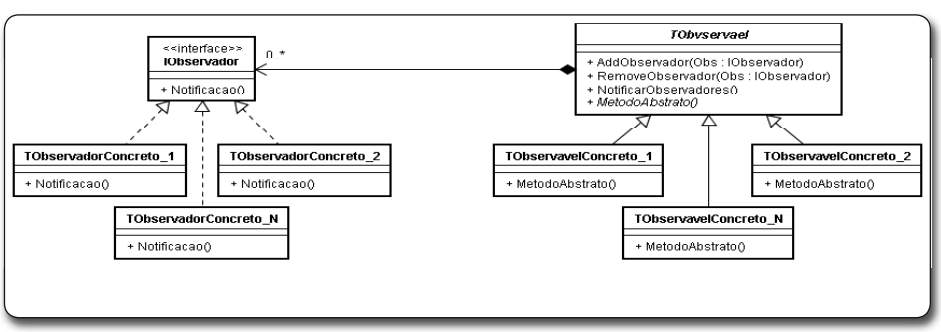


Figura 5. O novo componente do formulário

De volta aos Padrões

O problema atual do SendMail está relacionado com a criação (fabricação) dos objetos de transporte, pois a classe TFormulario deseja criar uma classe de transporte sem ao menos saber o nome dela. Já existe uma solução pronta, documentada, testada e ainda com a garantia de que os princípios de orientação a objetos serão seguidos. Esta solução tem um nome: padrão Factory.

O padrão Factory permite que uma classe crie outra sem ao menos saber como é o nome dela. Isto é possível devido ao uso de interfaces para definir grupos de classes. Sem o princípio "programe para interfaces" o padrão Factory seria apenas um sonho. As alterações para implementar este padrão são simples e rápidas. A **Figura 6** mostra como deve ficar o projeto SendMail depois que o padrão for implementado.

Como é possível notar, a classe TFormulario ficará dependente somente da classe TTransporteFactory (**Listagem 7**), e através dela poderá usar as classes de transporte sem ao menos seus nomes.

Agora a classe TFormulario finalmente está independente das classes de transporte, e mais uma responsabilidade foi encapsulada: a responsabilidade de criar as classes de transporte.

Com pode ser visto na linha 12 da **Listagem 7** o método Factory da classe TTransporteFactory recebe todas as informações necessárias para criar uma classe de transporte. Apesar disto algumas informações são ignoradas quando a classe TTransporteArquivoTexto for usada e como pode ver na linha 15 esta classe será usada sempre que o parâmetro Offline for True. Assim a classe TTransporteFactory consegue fabricar e devolver uma instância das classes de transporte para TFormulario sem que ela precise depender delas.

Com isto o método BotaoEnviarClick da classe TFormulario ficou mais simples ainda, no código a seguir é mostrado como é criada uma instância de ITransporte.

```
Transporte := TTransporteFactory.Factory
(ArmazenarLocalmente.Checked, SmtpHost.Text,
 Porta.Value, UsarSsl.Checked);
```

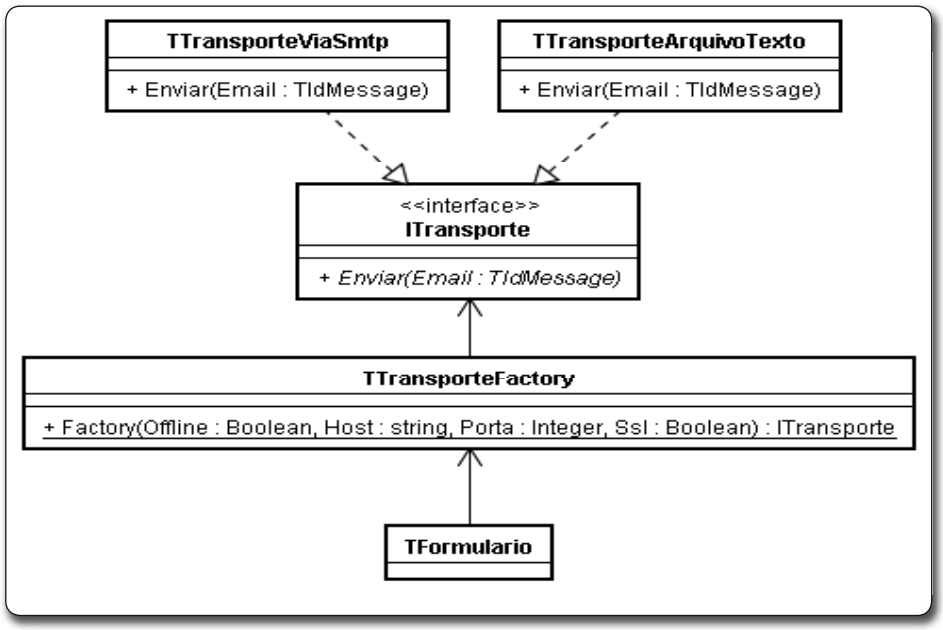


Figura 6. Projeto SendMail depois de implementar o padrão Factory

Listagem 7. Uma fábrica para as classes de transporte

```
01  unit TransporteFactory;
02  interface
03  uses Transporte;
04  type
05   TTransporteFactory = class
06   public
07     class function Factory(Offline: Boolean; Host: string;
       Porta: Integer;
08       Ssl: Boolean): ITransporte; static;
09   end;
10  implementation
11  uses TransporteArquivoTexto, TransporteViaSmtp;
12  class function TTransporteFactory.Factory
    (Offline: Boolean; Host: string;
13   Porta: Integer; Ssl: Boolean): ITransporte;
14  begin
15   if Offline then
16     Result := TTransporteArquivoTexto.Create('C:\Emails.txt')
17   else
18     Result := TTransporteViaSmtp.Create(Host, Porta, Ssl);
19  end;
20  end.
```

Conclusão

O padrão Strategy usa as interfaces (ou classes abstratas), para definir um grupo de objetos que encapsulam comportamentos diferentes. Assim é possível escolher qual comportamento usar em tempo de execução, dando mais flexibilidade ao projeto. Foi exatamente o que aconteceu com o projeto SendMail. Quando chegou a alteração pedindo para criar uma nova forma de transportar e-mails, observou-se que o comportamento de enviar e-mails mudava conforme a necessidade do cliente. Partindo desta informação a responsabilidade de enviar e-mails foi separada (encapsulada) em classes diferentes que se chamavam TTransporteViaSmtp e TTransporteArquivoTexto. Para qualquer um que conhece o arquivo padrão Strategy fica evidente a necessidade de utilizá-lo. Para aqueles que não conhecem o padrão, mas conhecem o princípio programe para interface, é quase intuitivo a ação de criar uma interface (ou classe abstrata) para definir o novo grupo de classes que encapsulam o comportamento de enviar e-mails.

Ao criar uma interface que define o grupo de classes que sabem transportar

Nota

Princípio de programação orientada a objetos – "Identifique os aspectos que mudam em seu aplicativo e separe-os dos que permanecem iguais."

e-mails, o padrão Strategy está consolidado, pois o Strategy se trata justamente de um uma interface definindo um grupo de classes que têm o mesmo objetivo, mas que o alcançam de maneira diferente. Trazendo para a realidade, no projeto SendMail, o padrão Strategy foi usado para definir um grupo de classes que sabiam transportar e-mail, mais que faziam de maneira diferente.

O projeto SendMail ainda pode se tornar mais flexível e além disto as alterações causaram a perda de um comportamento muito útil: a notificação de progresso da conexão. Na primeira versão do projeto, quando toda a lógica estava em um único método dentro da classe TFormulario, usava-se o evento OnStatus do objeto TIdSmtp para receber as mensagens de status da conexão. Porém agora fica difícil notificar o formulário do progresso da conexão, se o formulário nem ao menos sabe que a classe TTransporteViaSmtp existe. O padrão Observer pode ajudar na solução deste problema.

Rafael Stavarengo
faelsta@gmail.com
http://stavarengo.com
Programador de sistemas há 8 anos, integrante da equipe editorial da Revista ClubeDelphi. Domínio em Java, PHP e UML. Sólido conhecimento em Design Patterns e metodologias ágeis. Graduado em Análise e Desenvolvimento de Sistemas pela UNIPAR.

Blog do escritor Martin Fowler
http://martinfowler.com/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

WebMobile PLUS

A Revista **WebMobile** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/webmobile

Curso em .NET: **Aplicação completa de orçamento Doméstico no Visual Studio 2005**

Curso em Java: **Introdução ao uso de Web Services em Java ME**

A sua melhor opção de aprendizagem!
Assine a **WebMobile** e
Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

Faça a renovação de sua assinatura agora mesmo.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.